TILBURG UNIVERSITY

Tilburg University

Monitoring multi-party contracts for E-business

Xu, L.

Publication date:
2004

Document Version
Publisher's PDF, also known as Version of record

Link to publication in Tilburg University Research Portal

Citation for published version (APA):
Xu, L. (2004). *Monitoring multi-party contracts for E-business*. [Doctoral Thesis, Tilburg University]. CentER, Center for Economic Research.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 05. May. 2023

MONITORING MULTI-PARTY CONTRACTS FOR E-BUSINESS

Lai Xu

MONITORING MULTI-PARTY CONTRACTS FOR E-BUSINESS

Proefschrift

ter verkrijging van de graad van doctor aan de Universiteit van Tilburg, op gezag van de rector magnificus, prof.dr. F.A. van der Duyn Schouten, in het openbaar te verdedigen ten overstaan van een door het college voor promoties aangewezen commissie in de aula van de Universiteit op vrijdag 20 februari 2004 om 14.15 uur

door

Lai Xu

geboren op 4 maart 1970 te Xi'an, China

Promotor: prof.dr.ir. M. P. Papazoglou
Copromotor: dr.rer.nat. M. A. Jeusfeld

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems (Series No. 2004-02), at the Faculty of Economics and Business Administration of Tilburg University.

Copyright ©2004 by Lai Xu
All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any from or by any mean, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission from the publisher.
ISBN 90-5668-128-1

Abstract

Contracts between multiple business partners play an increasingly important role in a global economy where activities along the value chain are executed by independent, yet co-operating companies. Information technology to enact a value chain is now being deployed in the form of ERP systems and Web services. However, little is known about how to check formally whether such an enactment indeed fulfills the contract between the parties.

This dissertation investigates which parts of a contract can be formalized to be automatically monitored. The problem is addressed as a *formalization* problem: Given a paper contract, formalize it into suitable representations. Essentially, informal requirements (the paper contract) are mapped into formal specifications that are subject to automated processing – much in the same way system requirements are mapped into implementations.

Our approach supports not only the detection of actual violations, but also the pro-active detection of imminent contract violations. A paper contract is represented as a formal e-contract using temporal logic (a logic of propositions whose truth and falsity may depend on time). Such a formulation provides a possibility for pro-active monitoring. At the same time, we introduce our monitoring mechanism, which is designed to dynamically monitor our monitorable contract during the contract execution.

The multi-party contract is also explored. Monitoring a multi-party contract requires information from all participating sides. A failure of one party may lead to a follow-up failure of the performance of some other parties. The combination of all bilateral commitments is thus seen as part of a single multi-party contract. This integrated representation allows us to formulate clauses about "acceptable" or "required" behavior that range over more than two business partners.

To ensure receiving information from all participating parties, we also provide a framework for our monitorable contract model. We explain how this framework can be adapted to different e-commerce infrastructures and its flexibility for supporting different monitoring requirements.

Finally, we also provide a prototype, which was developed in Prolog.

Preface

This thesis is the result of my own work. The pronouns 'we' and 'our' in the text have been used for stylistic reasons.

Acknowledgments

The research leading to this thesis and the write-up have taken four years, during which many people have offered me intellectual and moral support. I would like to thank those who have loved and supported me. My family in China and the United States have offered so much support. Thank you.

I am grateful to my promotor, Prof. Mike Papazoglou, for his guidance and help. In retrospect, I am particularly amazed by how pleasant and patient Mike has been over the last four years.

Especially, I am extraordinarily grateful to my supervisor, Dr. Manfred A. Jeusfeld. Manfred has guided me patiently and continuously. He taught me what a Ph.D. thesis should look like, and how to dig a deep "hole" in my Ph.D. research; he explained what research questions are worth discussing and where to look for answers; he showed me how to write a research paper using common scientific languages, generously sharing his experience, ideas and insights with me. He encouraged me to develop my ideas, challenged me to improve them, and reassured me when I was upset – without ever complaining or losing his patience. If this thesis says anything useful or interesting it is very much due to Manfred – although I alone must take responsibility for any mistakes and misunderstandings. He also made it possible for me to finish my Ph.D. within four years. He has been the best supervisor I could possibly have hoped for and I am privileged to have been his student.

As a foreign student, I am also indebted to my former supervisor, Dr. Hans Weigand, the first Dutch person I had ever met, who picked up me at Schiphol airport when I arrived in a totally strange country. My thanks to him for his encouragement, enlightening explanations and discussions.

Very special thanks to the members of my Ph.D committee: Prof. Paul Grefen, Prof. Gerhard Lakemeyer and Prof. Barbara Pernici for their careful reading, thoughtful comments and corrections.

I enjoyed the warm energetic hospitality of Prof. Piet Ribbers, who taught me the art of positive thinking, which will notably benefit me for my whole life. I would especially like to thank Dr. Jian Yang for her support, experience, encouragement and critical insights during my Ph.D. studies. We shared many nice times, which I will always remember and enjoy.

I will be forever grateful to my friends and colleagues at Tilburg University for their warmth and friendship during these four years away from home. My thanks go to Drs. Bart Orriëns, the first reader of my thesis, for correcting my Chinese-English, for the exciting and heated discussions we had in mensa and the office, and for being a such great friend. To Marina V. Velikova, Ebru Angun, Mohammed Ibrahim, Amarendra Sahoo, Xiang Gao, Akos Nagy, and Sergei Artishchev: thanks for their wonderful company during these four years. I also owe a great debt to all my colleagues of Infolab, Department of Information Systems and Management, CentER research school and the Dutch research school for Information and Knowledge Systems (SIKS).

Thanks to my parents, I had a wonderful childhood. They guided me step-by-step to follow my dream of being a scientist. Because of their support I was able to pursue all of my interests in art, literature, philosophy and science. My name in the Chinese languages means "coming very slowly". My progress of choosing my research area has been slow, yet finally steady on Computer Sciences and Artificial Intelligence which were their research area as well.

Especial thanks to Drs. Paul T. De Vrieze for his love and support in personal life and academic research, particularly in the final year of the writing process. Many thanks to De Vrieze family for their support and care during my Ph.D. time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A decade ago, IT through its innovations in business process reengineering led the way in breaking down the inefficiencies within companies. Firms in the new millennia now face relentless pressure to perform better, faster, cheaper, while maintaining a high level of guaranteed results, etc. Firms must thus focus on their core business and outsource all other activities [BPM03]. Working with a partner, however, requires breaking down the inefficiencies between companies and coping with frequent change across the entire end-to-end value chain. In this new world of collaborative commerce and collaborative souring, a standard business process is simply inadequate. Using contracts to build new business relationships and to fulfill e-contract through Internet are important trends.

This chapter introduces the notion of e-contracts, contract life cycle and monitoring contract life cycle. Section 1.1 introduces the background to this research. Section 1.2 highlights the research motivation, requirements and issues. The goal and tasks that the research should achieve are listed in Section 1.3. Section 1.4 describes the contributions of research are described. This chapter ends with an outline of the structure of this thesis.

1.1 Research background

A contract records the agreed upon obligations of contractual parties in terms of business process conditions [WX01]. It identifies the parties' roles, responsibilities, obligations and deliverables [SSC+01]. It defines the set of activities, roles, and responsibilities to be taken by different parties to satisfy the terms and conditions in the contract. We will review the history of e-contracting from legal and technology aspects, respectively.

1.1.1 History of e-contracting

Although legal contracting is not a main concern in our research, it is an important part of e-contracting. We thus provide below a summary of Chapters 1, 2, and 6 of Daskalopulu's thesis [Das99].

> Over the last twenty years or so, a growing body of research in artificial intelligence has focused on the representation of legislation and regulations. In paper [Ser91], Sergot gave the long and established record of research that sought to apply artificial intelligence techniques to legislation. The idea of applying similar techniques to the representation of contracts is not new, and has in fact been emerging from time to time, as contracts serve a function similar to that of legislation: they are meant to regulate the actions of two or multi-parties while they interact.
>
> In 1987, Gardner [Gar97] concentrated on contract formation rules as her case study in developing a framework for the representation of legal rules informed by jurisprudence. Her work was still concerned with legislation about the nature of exchanges that lead to contractual relations, rather than legal contracts themselves.
>
> In 1992, The ALDUS project [Pro92] investigated the potential for developing systems to assist with the drafting of contracts, focusing on the Sale Goods contracts, which are relatively simple legal contracts. In 1997 and 1998, Yoshino report their work [Yos97], [Yos98] on representation of the United Nations Convention on contracts for the international Sale of Goods. Daskalopulu in her dissertation [Das99] explored the potential for developing logic-based tools for the analysis and representation of legal contracts.

The law regards contracts as collections of obligations. Research in this area includes automated inference methods, which are intended to facilitate application of the theory to the analysis of practical problems. The purpose of a legal e-contract system is to clarify and expand an incomplete and imprecise statement of requirements into a precise formal specification. Research thus mainly refers to deontic logic for formalization: duty, right, and other complex legal concepts.

Note that an e-contract in technology development has very different motivations and perspectives than an e-contract in legal exploration. In the early 1990's specialists created EDI, which was considered as a term that refers solely to electronic transactions and contracts [oJC95].

EDI requires an agreement between trading partners that not only dictates a standard data format for their computer-to-computer communications, but also governs all related legal issues of EDI usage. In 1987, the

first set of EDI rules was named Uniform Rules of Conduct for Interchange of Trade Data by Teletransmission (UNCID) [UNC87]. In 1990, the American Bar Association published a Model Trading Partner Agreement and Commentary together with an explanatory report, which were developed by the ABA's Electronic Messaging Service Task Force [WW01]. In 2000, IBM submitted to OASIS the first examples of XML-based EDI TPA (called Trading Partner Agreement Markup Language (tpaML) [DND+01]).

However, with the development of the Internet (which is regarded as a public network), electronic contracting began to be interpreted as a more broad term. E-contracts are also used across different workflow systems [KGV99], [KCK01], to cross different organizational business processes e.g. [CCT02], to integrate different web services [CCT03], [CCK+02], etc. E-contracts have become synonymous for business integration over electronic networks.

In papers [AG03] and [GA02], the authors described five e-contracting business processes and thus classified five e-contracting paradigms. It has a business process point of view look into e-contracting.

Legal e-contracting thus focuses on designing a contractual document to express as closely as possible the intention of the parties involved. Legal contract performance tools aim to advise parties on the effects of individual provisions, once an agreement is in force, to assist in planning the daily business exchange and to monitor the parties' compliance with the contract. Legal contracting also has a consulting function in contract performance. Technical e-contracting, on the other hand, focuses on business integration and automations. It is important to distinguish this difference between legal e-contracting and technical e-contracting. Chapter 2 reviews different logics and theories used for e-contracting, and relevant research in both types of contracting, to explain how to select suitable logics or theories for a particular e-contract application. The next section overviews contract definitions from different resources and presents the contract life cycle.

1.1.2 Contract definition and life cycle

We list the following definitions of contracts (including general definitions from dictionaries, definitions from the Laws of different countries, and a definition from general Law):

> *A contract is more or less an agreement entered into freely by a party with at least one other, to deliver goods or services, or to do something in return for some consideration (usually financial), on mutually agreed and binding terms, often in writing.* (Collins Dictionary)
>
> In English Law, *a contract is "an agreement which is legally enforceable or legally recognised as creating a duty".*

> In American Restatement Contracts, *"A contract is a promise or a set of promises for the breach of which the law gives a remedy, or the performance of which the law in some way recognizes as a duty."*
>
> The law views *contracts (agreements and their associate documents, where they exist) as entities that are created at a given point in time, persist over some specified period and then are extinguished (naturally by fulfillment, or unnaturally by early termination, as we shall see later).*

IBM's TPA (Trading Partner Agreement) is defined as an "electronic contract that uses XML to stipulate the general contract terms and conditions, participant roles (such as buyers and sellers), communication and security protocols, and business processes (such as valid actions and sequencing)" [DND+01]. There are new concepts of e-contracting from EDI which are closed e-contracting and open e-contracting. Closed electronic contracting can be defined as the use of EDI to expedite contracting among parties that already have trading relationships established. Open electronic contracting allows the formation of contracts among parties with no prior trading relationships, and is sometimes called "arm's length transactions" [Lee98b].

Our research emphasizes two important concepts for e-contracts. "contracts build a new business relationship between contractual partners", and "a contract is a guarantee". First, contractual partners build a business relationship using a contract such as an "arm's length transaction". Crossing workflow systems is a similar concept: two partners, who used different workflows, can cooperate by using e-contracts to support business automation [KGV99] [KCK01].

Second, the contract provides a guarantee to all contractual partners according to the clauses of the signed contract and relevant Laws. For example, Service Level Agreements provide a QoS for their parties [LKD+03] [KL03] that can be enforced. Another example is the contract used in the object-oriented programming language Eiffel (details can be found in Chapter 2). If the pre-conditions hold, the component guarantees certain post-conditions after the call. There exist some e-contracting applications that actually cover both sides' concepts. For instance, TPA in ebXML provides a new long-term business relationship. It also finishes a certain business exchange with a certain quality.

Generally a contract has the following stages [AG01][MAO96] [JFJ+96] [GSSS00] [Das99]:

- contract establishment or contract formation, which includes contract conception, preparation and negotiation activities, and
- contract fulfillment or contract performance, which is related to the

parties' behavior to the contract and may include monitoring, enforcement and compensation activities. This also includes contract finalization.

After having addressed e-contract concepts, we will proceed to discuss the contract life cycle. We are particularly interested in the contract monitoring life cycle at a contract fulfillment stage. This will be discussed in the next section.

1.1.3 Contract fulfillment monitoring life cycle

A *Contract Fulfillment Monitoring Life Cycle* is presented in Figure 1.1. We consider two monitoring stages: before anomalous actions occurrence and after anomalous action occurrence [Kle00] [KD01]. Before anomalous action occurrence, we can avoid and anticipate anomalous actions; based on the results of monitoring parties' activities, an enforcing mechanism ensures that the actual behavior conforms to the contract. After anomalous action occurrence, we need to detect and compensate anomalous actions, or store the unsolvable disputation for future human-involved resolution.

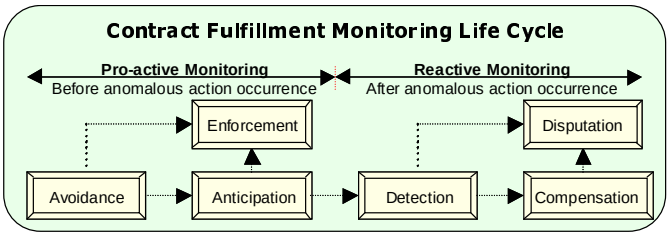


Figure 1.1: The contract fulfillment monitoring life cycle

This section introduces some background knowledge about the history of e-contracting, contract definitions, the contract life cycle, and the contract fulfillment monitoring life cycle. Our research motivation, requirements and issues will be explained in the next section.

1.2 Research motivation, requirements and issues

A considerable amount of recent research and industrial application effort has concentrated on the provision of standards for the automated establishment and subsequent implementation of electronic contracts. In general terms, a contract between multiple business partners contains some statements about their business relationship, particularly on their physical and informational interactions. One purpose of such a contract is to distinguish expected and acceptable behavior from behavior violating some clause in the

multi-party contract. Most of the current work focuses on the automation of contracting processes, rather than the development of services for contract fulfillment monitoring. We will proceed as follows, Section 1.2 provides the details of the research motivation, Section 1.2.2 specifies the research requirements, and Section 1.2.3 presents the research issues.

1.2.1 Research motivation

The introduction of workflow systems and enterprise resource planning systems increases the automation of business contract execution. To the same degree, the demand for automated monitoring increases because more information about the contract execution has to be processed by the business partners.

The most comparable work to this thesis can be found in studies of web service level agreements (WSLA) [LKD+03], [KL03], which are specialized agreements for guaranteed Quality of Service (QoS). This is, however, still rather far from our motivation. Mainly, we seek to improve monitorability of e-contracts when they are executed in e-commerce environments, not to particularly define an agreement for quality guarantees.

Traditionally, collaboration between business partners along a value chain are governed by bilateral contracts. A value-added provider of services contracted to multiple business partners would create a collection of such bilateral contract. As we see later in this thesis, monitoring such a complex collection of agreements requires information from all participating sides. A failure of one side of some bilateral contract may lead to a follow-up failure of some other partner standing is another bilateral contract. Hence we view the combination of all bilateral commitments as part of a single multi-party contract. This integrated representation allows to formulate clauses about "acceptable" or "required" behavior that range over more than two business partners.

In addition, little research has been done on multi-party contracts. Basically all research [Hau02], [Dub02] on multi-party contracts tries to break down a multi-party contract into a number of bilateral contracts. In some cases, it is possible to do so without losing valuable information. However, as more multi-party relations will exist between companies, more contracts will be in force that would result in loss of information and increased complexity as relationships get hidden. The reason why most research into multi-party contracts try to break down multi-party contracts into a few bilateral contracts, is that they try to adapt e-commerce environments, which can only support bilateral environments, to support a multi-party contract execution. In other words, these approaches attempt to execute a multi-party contract in a bilateral way. They assume the whole business process will going OK according a number of bilateral contracts. In so doing, these studies ignore a big issue, which is contract violation. In multi-party contracting, it is more

difficult to find the responsible party (or parties) for a contract violation. Although retrieval of all bilateral contracts would assist in the identification of a responsible party (or parties) for a contract violation, the issue is more complex because of the loss of information that occurs under the transformation from a multi-party contract to a number of bilateral contracts. Our other concern is thus the multi-party contract fulfillment monitoring.

Accordingly, our motivation is to explore monitorability of e-contracts in general and to focus on the multi-party contract monitoring at the contract fulfillment stage.

1.2.2 Research requirements

As the monitoring contract fulfillment life cycle was described in Section 1.1.3, new monitoring requirements can be noted from two perspectives: the *pro-active monitoring perspective* and the *reactive monitoring perspective*.

There are three monitoring requirements from the pro-active monitoring perspective:

1. Contractual parties need to be monitored for the purpose of avoidance and anticipation. Non-performance action needs to be enforced to execute.

2. The execution of the actions needs to be measured to assure performance qualities.

3. Relevant events need to be recorded. After conflicts between contractual parties, these records can be used as evidence of what actually happened and who is responsible.

The monitoring requirements from the reactive monitoring perspective may be elaborated as follows:

1. Anomalous actions need to be detected. Especially in multi-party contractual business processes an anomalous action can sometimes be detected only after other parties have performed many actions. Retrieval of certain activities of different parties is necessary.

2. The non-conforming actions or anomalous actions need to be compensated. Sometimes the compensation function is optional, but the other parties must at least be informed of the detection of anomalous actions to prevent further cost.

3. Unsolvable disputations need to be stored for future human-involved arbitration and resolution.

Following our research requirements for pro-active monitoring and reactive monitoring purposes, the research issues are introduced in the following section.

1.2.3 Research issues

In accordance with our research motivation and requirements, our research is aimed at improving monitorability of multi-party e-contracts at the contract fulfillment stage. Our research concentrates on monitoring the execution of contracts. The monitoring is a service to the business partners that shall be used to improve their performance with respect to contract requirements. In general, the research issues include

1. How to specify a formal model of e-contract computations to give a solid foundation for the reasoning necessary of monitoring e-contracts?
   - Which elements should be included in the contract model to represent the "fact" part of a contract?
   - Which elements should be included for reasoning the process of the contract execution?
2. How to dynamically schedule actions to achieve the pro-active monitoring?
   - Which kinds of dynamic mechanisms can be used at the contract fulfillment stage?
3. How can our contract model and dynamic mechanism be used at existing e-market environments?
   - Which kinds of the e-market infrastructures are there?
   - How to integrate our contract model and mechanism into existing e-markets?

Each of the different monitoring stages features it own concrete research questions and statements of purpose. The pro-active monitoring stage features two monitoring functions that should be carried out by our monitorable contract:

1. Given the current state of contract execution, which actions are expected from a partner in the future?
2. Is a contract violation likely to happen within a short period of time? Which partners must be reminded to fulfill their obligations?

At the reactive monitoring stage there are two monitoring functions:

1. Which partner is responsible for a contract violation?

Based upon the above, a complete contract monitoring process should be able to perform the following functions:

- To avoid anomalous actions,

- To anticipate imminent contract violations,
- To enforce non-conforming actions at the pro-active monitoring stage,
- To detect contract violation, and
- To find out who is the responsible partner for a contract violation.

We address the problem as a *formalization* problem: Given a paper contract, formalize it into suitable representations such that the above questions can be answered. Essentially, we map informal requirements (the paper contract) into formal specifications that are subject to automated processing very much like system requirements are mapped into implementations.

This section has explained our research motivation, presented research requirements and summarized our research issues. The following section specifies our research goal and tasks.

1.3 Research goal and tasks

Our research concerns a range of contract-based business automations, exploring particular the monitorability of e-contracts. The *research goal* has been the development of a new contract model to conveniently monitor multi-party contracts at the contract fulfillment stage.

Research tasks are specified as follows:

- Formalization of the monitorable contract model.
- Representation of multi-party contracts.
- A new framework within which our monitorable contract model can run.
- Prototype implementation and performance tests under different workloads in order to estimate the extra computational costs exerted by the monitoring component on an e-commerce system.

1.4 Contributions

This thesis investigates the monitorability of e-contracts–e.g. which parts of a contract can be formalized to enable automatic monitoring. Subsequently, we propose a new contract model that allows for the convenient monitoring of multi-party contracts during contract fulfillment and provides pro-active monitoring functions.

We use logic relationships between every action or activity to deal with monitoring issues, which makes it possible to achieve the pro-active monitoring goal. Chapter 2 analyzes related work from monitoring issues in

event-based systems, to e-contract related logics, current contract models or languages, and monitoring architectures. We show that our research is unique and original in pro-active monitoring using temporal logic.

Little research has been done on multi-party contracts [Hau02], [Dub02]. Basically all research on e-contracts up to this point tries to break down a multi-party contract into a number of bilateral contracts. In some cases, it is viable to do that. However, as more multi-party relations will exist between companies, more contracts will be in force that would result in loss of information and increased complexity as relationships get hidden. We use a car insurance case (details can be found in Appendix A) to explain why a multi-party contract can not be separated into a few bilateral contracts. We present our commitment graph to model a multi-party contract that will help contractual parties to negotiate an enforceable contract at the contract establishment stage, and also to find a responsible party (or parties) for a contract violation at the contract fulfillment stage.

In short, our contributions can be summarized as follows:

- We provide the pro-active monitoring concept for contract monitoring [XJ03], [Xu03b], [Xu03a];
- We present a formal model of contracts [XJ03], [Xu03b];
- We show a multi-party contract modeling tool and its specifications [XJ03];
- We improvement of monitorability in general [XJ03], [Xu03b], [Xu03a].

1.5 Dissertation outline

The main body of this dissertation is organized as follows:

Chapter 2 reviews related work from different dimensions (including broad views from multiple-disciplines' monitoring approaches, from contract-related logics and theories, from contract models and languages, and from contracting frameworks or architecture). For each of these, the weaknesses and limitations are analyzed and highlighted. Our analysis provides us with an orientation point within the literature for this research.

Chapter 3 presents our propositional temporal logic, which forms the part core of our monitorable contract model. This chapter provides a formal syntax and semantics of propositional temporal logic, and proves propositional temporal logic.

Chapter 4 presents a formal model of monitorable e-contracts, which is another vital part of our research. The static contract model includes contract elements and logic relationships. In our model, a contract consists of actions and commitments. We use our propositional temporal logic to represent logic relationships between the actions (called contract constraints). A

guard of a contract constraint dynamically tracks the contract performance state.

Chapter 5 is concerned with the monitoring mechanism, which is used in our monitorable contract model. We derive a dynamic monitoring mechanism based on the static monitorable contract model. We also explain the commitment graph, maintaining guards algorithm and pro-active detection algorithm. These work together to enable the monitoring functions discussed in Section 1.2.3.

Chapter 6 introduces a framework within which our contract model can run. We explain how this framework can be adapted to different e-commerce infrastructures and demonstrate its flexibility for supporting different monitoring requirements.

Chapter 7 outlines the prototype implementation and discusses related techniques.

Finally, Chapter 8 summarizes the findings of this research and discusses directions for future work.

Chapter 2

Related Work

In Chapter 1, we summarized e-contracting history. This chapter deeply investigates related work from different perspectives. Section 2.1 discusses multi-disciplinary monitoring approaches. Section 2.2 looks into contract related logics and theories. Section 2.3 reviews existing contract models and languages. Finally, Section 2.4 presents contract related frameworks and architectures.

2.1 Multi-disciplinary monitoring approaches

Monitoring issues are widely discussed in many disciplines. This section investigates contract-related monitoring approaches in different research areas for different purposes. In Section 2.1.1, contracts are used in object-oriented programming language for developing reliable software. In Section 2.1.2, contract representation and assessment in the area of Artificial Intelligence give a totally different perspective. Section 2.1.3 discusses various monitoring approaches in multi-agent systems. In Section 2.1.4, event-based monitoring also adds some useful values to our monitoring mechanism. As mentioned in the previous chapter, our concern is pro-active monitoring of multi-party contracts at the contract fulfillment stage, which is a new application in business process automation. This chapter explores and compares a broad range of technologies and formalization, together with some of the foundations upon which this thesis is built.

2.1.1 Programming languages

Regarding the object-oriented constraints perspective, Meyer [Mey97] [Mey] refined the assertion-based approach into the design-by-contract method in the Eiffel language. The basic idea is that a component and its clients have a contract with each other. The client guarantees certain preconditions before calling a method; the component guarantees certain postconditions after

the call. If the pre- and postconditions are included in a form that can be compiled, then any violation of the contract between caller and component can be detected immediately. The prime focus of the approach is to deliver reliable software, and can not, as such, include pro-active monitoring.

The idea of programming language using contracts to guarantee certain results is the same as when we want to guarantee that each contract has been compliantly fulfilled. However, the way to specify the contract and the way to detect contract violations differ completely. The next section reviews contract research in AI which also gives a different perspective in dealing with contract-related issues.

2.1.2 Artificial intelligence

Over the last twenty years or so, a growing amount of research in Artificial Intelligence has focused on the representation of legislation and regulations. Contracts as legal entities have been explored from different views: representation, reasoning [LR95] [Ser01], and assessment [DDM01] [DM01] [BLWW95].

In paper [Gar97], Gardner aimed to "create a model for the legal reasoning process that makes sense from both jurisprudential and AI perspectives". Her research concentrated on contractual offer and acceptance. To this end, she proposed a system that not only aims to solve legal problems, but also "to recognize the issues a problem raises and to distinguish between those it has enough information to resolve and those on which competent human judgments might differ".

Allen advocates through a series of paper [All80], [All82], etc. the use of symbolic logic as a tool for analyzing and interpreting legal text. His research concentrates on the use of logic to improve the language of the Law, by considering inadvertent ambiguity that arises in written legislative text.

As legislative and regulatory statements aim to direct human behavior primarily by specifying permissible, obligatory or forbidden actions, deontic logic (a branch of modal logic [vW51] that is concerned with norms and normative behavior), is a natural candidate for representing and reasoning with such statements. Deontic Logic finds its origins in Ethics and Legal Philosophy, but has more recently found applications in computer science and Artificial Intelligence, for example, as a means of specifying constraints of security policies [MW93a] and contracts [WX01].

Papers [Das99], [DDM01], [DM01] and [DTM02] works on assessing the status of legal contracts. Business procedures are based on a Finite State Machine, or Petri Net. Subjective Logic is used to evaluate the uncertainty of different parties' belief regarding the evidence-based contract performance monitoring. More details can be found in Section 2.2.4.

AI research in contracts focuses on legal reasoning, contract representa-

tion, contract specification, and contract assessment. Gardner and Allen's research, monitors whether contracts or legal texts are consistent through a legal process. Daskalopulu's research explores the contract performance monitoring issue, but her research mainly focuses on a legal view (evidence-based monitoring). This is an important issue, but our focus here is on contract automation monitoring from an IT perspective.

2.1.3 Fault-tolerance and monitoring issues in multi-agent systems

In dynamic multi-agent systems, agents must monitor their peers and the environment to execute individual and group plans, to ascertain their progress and to detect/tolerate failures. This section reviews several monitoring or tolerant approaches in various multi-agent systems, and analyzes the differences between these approaches and ours.

Hägg uses external sentinel agents to monitor inter-agent communication, build models of other agents, and take corrective actions [Hag96]. The sentinel-based approach detects inconsistencies by observing inter-agent behaviors. In contract fulfillment monitoring, inter-agent actions and those of external agent actions are all concerned with different business processes.

Klein proposes use of an exception-handling service to monitor the overall progress of a multi-agent system [KD99]. The exception-handling service is a centralized approach, whereas our contract fulfillment monitoring supports both centralized and decentralized monitoring.

Kaminka and Tambe use a social diagnosis approach wherein socially similar agents compare their own state with that of other agents in order to detect possible failures [KT98]. Although the socially-attentive monitoring approach is an explicit teamwork model, it does not provide the pro-active monitoring that our approach does.

Kumar and Cohen advocate re-arranging brokers when an agent that was registered becomes unavailable [KC00]. This technique is implemented by adding a plan to the plan library of a generic agent. It is an efficient way for multi-agent systems, but it is not realistic for contract fulfillment monitoring, which is not about recovering from broker failures, but about handling intentional misbehavior.

In general, multi-agent system fault-tolerant approaches analyze the entire communication in the system in order to detect state inconsistencies using replication strategies [MSBG01], sentinel approaches [Hag96], re-assignment resource approaches [KC00], and knowledge-based approaches [KD99]. Most of this research focuses on the infrastructure level. Our contract monitoring fulfillment focuses on a semantic level of monitoring to prevent imminent contract violation.

2.1.4 Monitoring issues on event-based systems

In event-based systems, the event notification service can carry out a selection process to determine which of the published notifications is of interest to which of its clients, routing and delivering notifications only to those clients that are interested. More specifically, the event notification service may be asked to apply a filter to the contents of event notifications, such that it will deliver only notifications that contain certain specified data values. The selection process may also be required to look for patterns of multiple events, such that it will deliver only sets of notifications associated with that pattern of event occurrences. This section reviews some relevant systems that are used in workflow management systems(WFMS) and Web systems.

Paper [MSS97] presents an interpreted generalized event monitoring language (GEM). It allows high-level, abstract events to be specified in terms of a combination of lower-level events from different nodes in a loosely coupled distributed system. GEM specifies the operation of event monitors. Each monitor contains a command interpreter, and can be controlled interactively by sending it the appropriate GEM scripts. A GEM script declares event classes, rules that define the actions to be taken when an event is triggered, and commands to trigger an event, to disable or enable rules etc. GEM is a declarative rule-based language in which the notion of real time has been closely integrated and in which various temporal constraints can be specified for event compositions.

SINEA [CRW01], [CRW98] is a scalable event notification service that is based on a distributed architecture of event servers. SINEA extends the familiar publish/subscribe protocol with an additional interface function called *advertise*, a function *unsubscribe* and a function *unadvertise*. SINEA adopts a peer-to-peer topology, a hybrid of the two structures–whether a hierarchy of peers, or peers of hierarchies.

CEA (Cambridge Event Architecture) supports asynchronous operation by means of events, event classes, and every occurrence as an object instances. CEA follows a publish-register-notify paradigm with event object classes and source-side filtering based on parameter templates [BMB+00]. Storage and query facilities for events are advocated to adequately support event-driven applications. In this architecture, contracts between domain can be created and used for event translation [BHM+00] [BMY03]. The contract in this architecture is similar to the external schema from a database federation point of view. In this way heterogeneous systems can be used together in a federation for tracking and analyzing events across multiple application domains.

EVE [GT98] is an event engine that implements an event-driven execution of distributed workflows. Its functionality includes event registration, detection and management, as well as event notification to distributed autonomous, reactive software components, which represent workflow process-

ing entities. EVE also maintains a history of all event occurrences in the system used for the monitoring and analysis of execution workflows.

JEDI (Java Event-based Distributed Infrastructure) [CNF01] is an object-oriented infrastructure that supports the development and operation of event-based systems and has been used to implement the OPSS workflow management system.

Le Subscribe [PFL+00] is an event notification system for the Web to deal with highly dynamic Web information. Another event notification system, READY system [GKP99], has a more expressive subscription language supporting grouping constructs, compound event matching and event aggregation. Its matching algorithm uses only local optimizations, unlike Le Subscribe, which intensively exploits global optimization opportunities. Paper [Hin03] concentrates on the filtering of composite events, which are formed by temporally combined primitive events.

Monitoring is particularly essential for all aspects of management of communication networks and distributed systems. Languages, which are used to specify events, filters, and patterns, primarily support event detection and notification distribution.

In papers [Abr02b], [AB00], [AB01a], [AB01b], [AB01d], [AB01c], [AB02b], and [AB02a], Abrahams aims to provide "a human analyst with sufficiently detailed methods to guide the interpretation of the specification and facilitate ...". The work explores the practical execution of business processes following contracts, policies and legal requirements. Specifically it proposes various types of queries that can be explained and stored using occurrences, which are triggered automatically by the system in accordance with the policies defined in the contracts (specifications) in the occurrence store.

Active databases have generally adopted Event-Condition-Action rules. Those rules can be used to specify different actions when a given condition is satisfied, depending on which event occurred [WC96]. AI rules languages and deductive database normally use rules without events, which are Condition-Action rules. In active and deductive databases, the events and conditions or only the conditions are evaluated to determine whether the actions occur. In our contract monitoring research, we try to look into logic relationships between actions which means after which action has occurred, which action can be expected. Thus, we reason about logic relationships between the actions, not logic relationships between the events, the conditions and the actions.

This section reviews different monitoring approaches from different perspectives. Most application domains are in the direction of the system administration, but not business process automation. The different domains have different requirements for monitoring. In our application, the business process automation requires semantic level monitoring, rather than system-level monitoring. However, some research results, ideas and approaches from different perspectives can be used for reference in our monitoring research.

The next section reviews contract-related logics and theories.

2.2 Contract-related logics and theories

Logic is an important tool in the analysis and presentation of arguments [Kow79]. Logic is a likely possible candidate for analyzing formal aspects of contract-related reasoning, since it is the very essence of logic to systematize formal patterns in reasoning. Logic is an obvious candidate for modeling the separation of knowledge and the ways of using it, because in logic this separation is total in the form of premises in some formal language on the one hand, and an inferential apparatus on the other. In short, logic is, at least at first sight, highly relevant for contract representation, assessment and monitoring.

This section reviews where logics may be used in contract-related issues, and explains what kind of logics could solve which kind of contract-related problems. Section 2.2.1 begins by discussing classic logics - *predicate logic*, *first-order logic*, and *speech act theory*. Next, *deontic logic* is presented in Section 2.2.2, *temporal logic* is described in Section 2.2.3, and *subjective logic* in Section 2.2.4.

2.2.1 Predicate logic, first-order logic and speech act theory

Predicate logic is a branch of logic that deals with propositions in which subject and predicate are separately signified, reasoning whose validity depends on this level of articulation, and systems containing such propositions and reasoning. First-order predicate logic is a Predicate logic in which predicates take only individual arguments, and quantifiers bind only individual variables. They are well-known branches of logics. There is no example of logic-based contract models that refers to only predicate logics or first-order logic. However, almost all logic-based contract models somehow use them – for example, Lee's logic model for e-contracting [Lee98a], or Weigand and Xu's contract model [WX01].

Although Speech Act theory is not about logic, it has been widely used with various logics to express the logic-based contract model or business communication. Part of Austin's work, Speech Act theory [Aus76], is the observation that utterances are not implied propositions that are true or false, but attempts on the part of the speaker that succeed or fail. The performatives, acts, or actions are organized as speech acts and non-speech acts. An individual speech act is either a solicit, which explains an attempt to achieve mutual belief with the addressee that the sender wants the addressee to perform an act relative to the sender's wanting it done, or an assert, which expresses an attempt to achieve mutual belief with the addressee that the asserted statement is true. Each of these can be refined on the basis of the kind of action that the sender is soliciting or the nature

of the proposition that the sender is asserting, respectively, as suggested in Figure 2.1.

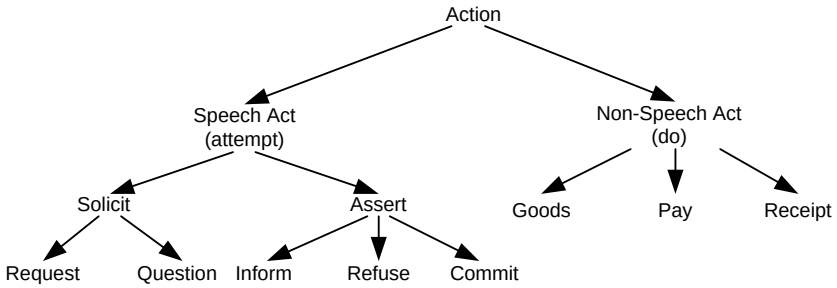


Figure 2.1: Base types of communication acts [Par96]

Kimbrough and Moore formalize the speech act theories and apply these ideas to deontic reasoning [KM93] and business messaging as Formal Language for Business Communication (FLBC) [KM97], [Moo00]. We use Speech Act theory related work [Par96] to create our commitment graph (details can be found in Chapter 4) in our research. Predicate logic, first-order logic, and speech act theory are fundaments of the logic-based contract model. The next section introduces deontic logic and its application in e-contracting.

2.2.2 Deontic logic

Whereas Speech Act theory describes the acts or actions of the contract, deontic logics studies the nature of obligation, which refers to whether an action is obligatory, and not whether it occurs. Particularly, from a legal point of view, contracts primarily aim to direct contractual parties' behavior by specifying permissible, obligatory and forbidden actions.

Deontic logic is the study of the logical relationships among propositions that assert that certain actions or states of affairs are morally obligatory, morally permissible, morally right or morally wrong. The initial proposals were derived from Von Wright [Wri68]. As a basic concept, he introduced the following operations: O stands for 'obligatory', P stands for 'permitted', and F stands for 'forbidden'. Different variations of deontic logic continue to be proposed and debated. The core of current developments in Deontic Logic concerns the *standard system of deontic logic* [Che80]; we summarize its axiom and rules as follows [MW93b]:

Axioms

(KD0) All (or enough) tautologies of propositional logic
(KD1) $O(a \rightarrow b) \rightarrow (Oa \rightarrow Ob)$
(KD2) $Oa \rightarrow Pa$

Rules

(KD3) $Pa \leftrightarrow \neg O \neg a$
(KD4) $Fa \leftrightarrow \neg Pa$
(KD5) $a, a \rightarrow b \vdash b$
(KD6) $a \vdash Oa$

The theory of Normative Positions developed by Sergot [Wie98] is a combination of deontic logic and the logic action/agency to the formalization of Hohfeld's [Hoh13] "fundamental legal conceptions", which are about "right", "duty" etc. From the e-contracting perspective, we list some related applications as follows: Lee's logic model for electronic contracting [Lee98a], Daskalopulu et al.'s evidence-based contract monitoring [DTM02], Weigand and Xu's contract model [WX01] and Ludwig and Stolze's Simple Obligation and Right Model (SORM) [LS03]. The next section explains how temporal logic can be used in e-contracting.

2.2.3 Temporal logic

Contracts specify one or more actions to be performed by the contractual parties. A sequence of actions is stipulated in contracts. Thus, temporal relationships are key to a logic of contracts, in order to deduce who is to do what, when, and what consequences apply if any parties fail to fulfill their obligations.

Temporal logic is a logic of propositions whose truth and falsity may depend on time. Closely related to modal logics, it has long been a matter of research. Precise formal foundations of various kinds of temporal logic have been developed during the last 30 years. We introduced a temporal logic of 'Axiomatization of Propositional Temporal Logic' in [Kro87], whose axioms and rules are summarized below, where $\bigcirc A$ means " A holds at the time point immediately after the reference point" , $\Box A$ means "A holds at all time points after the reference point", $\diamond A$ means "There is a time point after the reference point at which A holds", and A **atnext** B means "A will hold at the next time point that B holds".

Axioms
(taut) all tautologically valid formulas,
(ax 1) $\neg \bigcirc A \leftrightarrow \bigcirc \neg A$,
(ax 2) $\bigcirc(A \rightarrow B) \rightarrow (\bigcirc A \rightarrow \bigcirc B)$,
(ax 3) $\Box A \rightarrow A \wedge \bigcirc \Box A$,
(ax 4) $\bigcirc \Box \neg B \rightarrow A$ **atnext** B,
(ax 5) A **atnext** $B \leftrightarrow \bigcirc(B \rightarrow A$ **atnext** $B)$.

Rules
(mp) $A, A \rightarrow B \vdash B$,
(nex) $A \vdash \bigcirc A$,
(ind) $A \rightarrow B$, $A \rightarrow \bigcirc A \vdash A \rightarrow \Box B$.

Lee's logic model for electronic contracting refers to Von Wright's [Wri65] temporal logic and RU calculus [RU71]. Our monitorable contract model

utilizes our own proposition temporal logic to facilitate pro-active monitoring [Xu03b], [XJ03]. Whereas deontic logics are important for legal views on e-contracting, temporal logics are especially important for business automation aspects of e-contracting. The next section introduces another logic that is used in event-based contract monitoring.

2.2.4 Subjective logic

In the contract fulfillment stage, each contractual party has a different view of its own behavior and that of the counter-parties particularly with regard to whether they comply with the agreed contract. In standard logic, propositions are considered to be either true or false. However, subjective logic addresses the problem of forming a measurable belief about the truth or falsity of an atomic proposition asserting a property of the world, and uses the term *opinion* to denote the representation of a subjective belief. Subjective logic can be seen as an extension of both probability calculus and binary logic [Jös01].

An observer's opinion about a proposition x is a representation of a belief and is modeled as a triple $\omega(x) =< b(x), d(x), u(x) >$, where:
$b(x)$ measures belief, represented as the subjective probability that a proposition x is true
$d(x)$ measures disbelief, represented as the subjective probability that a proposition x is false;
$u(x)$ measure uncertainty, represented as the subjective probability that a proposition x is either true or false;
$b(x), d(x), u(x) \in [0 \cdots 1]$ and $b(x) + d(x) + u(x) = 1$, for any proposition x.

Subjective Logic Operators Various operations can be applied to atomic opinions to define compound ones. We lists some operations as follows:
Conjunction: Let $\omega_x =< b_x, d_x, u_x >$ and $\omega_y =< b_y, d_y, u_y >$ be an observer's opinions about x and y. Let $\omega_{x \wedge y} = (b_{x \wedge y}, d_{x \wedge y}, u_{x \wedge y})$ be the opinion such that

$$
\begin{aligned}
b_{x \wedge y} &= b_x b_y \\
d_{x \wedge y} &= d_x + d_y - d_x d_y \\
u_{x \wedge y} &= b_x u_y + u_x b_y + u_x u_y
\end{aligned}
$$

Disjunction: Let $\omega_x =< b_x, d_x, u_x >$ and $\omega_y =< b_y, d_y, u_y >$ be an observer's opinions about x and y. Let $\omega_{x \vee y} = (b_{x \vee y}, d_{x \vee y}, u_{x \vee y})$ be the opinion such that:

$$
\begin{aligned}
b_{x \vee y} &= b_x + b_y - b_x b_y \\
d_{x \vee y} &= d_x d_y \\
u_{x \vee y} &= d_x u_y + u_x d_y + u_x u_y
\end{aligned}
$$

Discounting: Let A and B be two observers, where $\omega_B^A = (b_B^A, d_B^A, u_B^A)$ is A's opinion about B's advice, and let x be a proposition where $\omega_x^B = (b_x^B, d_x^B, u_x^B)$ is B's opinion about x expressed in an advice to A. Let $\omega_x^{AB} = (b_x^{AB}, d_x^{AB}, u_x^{AB})$ be the opinion such that

$$\begin{aligned} b_x^{AB} &= b_B^A b_x^B \\ d_x^{AB} &= b_B^A d_x^B \\ u_x^{AB} &= d_B^A + u_B^A + b_B^A u_x^B. \end{aligned}$$

Then, ω_x^{AB} is called the discounting of ω_x^B by ω_B^A, expressing A's opinion about x as a result of B's advice to A. By using the symbol '$\otimes$' to designate this operator, then $\omega_x^{AB} \equiv \omega_B^A \otimes \omega_x^B$.

Consensus: Let $\omega_x^A = (b_x^A, d_x^A, u_x)$ and $\omega_x^B = (b_x^B, d_x^B, u_x^B)$ be opinions held by observers A and B respectively about the same proposition x. Let $\omega_x^{A,B} = (b_x^{A,B}, d_x^{A,B}, u_x^{A,b})$ be the opinion such that

$$\begin{aligned} b_x^{A,B} &= (b_x^A u_x^B + b_x^B u_x^A)/\kappa \\ d_x^{A,B} &= (d_x^A u_x^B + d_x^B u_x^A)/\kappa \\ u_x^{A,B} &= (u_x^A u_x^B)/\kappa \end{aligned}$$

where $\kappa = u_x^A + u_x^B - u_x^A u_x^B$ such that $\kappa \neq 0$. By using the symbol '$\oplus$' to designate this operator, we define $\omega_x^{A,B} \equiv \omega_x^A \oplus \omega_x^B$.

In conventional contracting, practice disputes that might arise between parties are normally resolved by presenting relevant evidence concerning factual aspects of the transaction to a commonly accepted arbitrator. In [DM01], [DDM01] and [DTM02] Daskalopulu et al. explored evidence-based contract monitoring. Each contractual party has different views about whether its own behavior and that of the counter-party complies with the agreed contract. Subjective Logic is used to measure and reason opinions of possible recommendations from the parties. They introduced two scenarios. Scenario 1 allows the contractual party to receive external advice from the central controller about the state of contract execution. Scenario 2 provides a self-regulated e-market in which all contractual parties automatically update their information about the state of contract execution.

2.2.5 Petri net and finite state machines

Also worth mentioning are, Petri Net and Finite State Machine, which were used to represent the events and actions of contracts and state of contracts respectively in Lee's work [Lee98a] and Daskalopulu's work [DDM01], [DTM02].

Section 2.2 summarized contract-related logics and theories that refer to legal and business automation aspects. Although introduced independently, these theories are becoming increasingly interconnected within the field of electronic contract execution.

2.3 Contract models and languages

This section reviews some contract-related work from the perspective of business processes in Section 2.3.1, and existing contract models and languages, in Section 2.3.2. Our main concern is their monitorability. Monitorability refers to whether sufficient information and monitoring points of a contract model are provided, so that the contract can be effectively monitored.

2.3.1 Business process languages

The Business Process Modeling Language (BPML)[BPM01] is representative of a new family of process definition languages intended for expressing abstract and executable processes that address all aspects of enterprise business processes. This includes those areas important for web-based services. Microsoft's XLANG [Tha01] and IBM's Web Services Flow Language (WSFL) [Ley01] are other members of this family. XLANG and WSFL have now been combined in Business Process Language for Web Services (BPEL4WS) [CGK+02]. BPEL4WS provides a language for the formal specification of business processes and business interaction protocols [CGK+02].

The WfMC (Workflow Management Coalition) proposed standard for an XML-based Process Definition interchange Language (XPDL) [WfM02], which is a standard of supporting the creation and processing of document classes. It provides documentation, constraint checking, attribute defaulting, entity replacement, style specification, and application extensions. In paper [Sha02], Shapiro compared BPML, BPEL4WS and XPDL from the perspective of business process execution.

The Web Service Choreography Interface (WSCI) is an XML-based interface description language that describes the flow of messages exchanged by a web service participating in choreographed interactions with other services. In paper [vdADtHW02], Van der Aalst et al. reported the difference between BPML and WSCI, based on pattern analysis.

Business process management faces the establishment of standards for process design, deployment, execution, maintenance and optimization, for which IBM's WSFL, Microsoft's XLANG, BPEL4WS, WSCI and XPDL provide a comprehensive structure for describing a business process in detail from different dimensions of the business process. Furthermore, they provide the possibility of describing assertions. Exception handlings are relevant to our monitorability of business processes. However, the above mentioned standards focus on non functional or procedural for monitoring or pro-active monitoring rigorously, neither provides any explicit support for the "business contract" level of abstraction. Our approach can be used to express of business flow relationship that improve a monitorability when a business flow run in a real time.

The next section continues to examine other existing contract models or

languages to observe how they deal with monitoring issues.

2.3.2 Existing contract models and languages

CrossFlow[KGV99] and **E-ADOME**[KCK01] use contracts for inter-organizational workflow process integration. Contracts in CrossFlow and E-ADOME describe the agreed workflow interfaces as activities and transitions, based on WfMC's WPDL (Workflow Process Definition Language). CrossFlow contracts [KGV00] define all data, process elements and enactment conditions relevant to the co-operation through the outsource workflow process on an abstract level. Contracts in CrossFlow and E-ADOME also specify what data objects in the remote workflow are readable or updateable. They are a side effect of business automations, and do not yet have pro-active monitoring capabilities.

Other Contract Models and Languages, such as Lee's e-contracting logic model [Lee98a], aim to improve both expressiveness and inferential capabilities of contracts; Weigand and Xu's contract model [WX01] focuses on task allocations and process co-ordinations. These tasks do not include contract monitoring.

Reeves et al.'s declarative language for negotiating executable contracts can facilitate modification during negotiation [RGWC99], and includes prioritized conflict handling features. However, it cannot be expected that all possible conflict scenarios throughout all business processes are recognized completely before the contract fulfillment stage. In **SeCo** (secure electronic contracts) [GSSS00], the monitoring services allow events to be triggered according to the current state of the contract, and they alert an enforcement service to allow it to initiate an enforcement activity. But SeCo looks only at the negotiation stage.

In existing e-commerce frameworks, such as the XML-based **Trading Partner Agreement Markup Language (tpaML)** from IBM research [DND+01], 'trading partner agreement' is used as 'contract'. tpaML is now pursued under the **OASIS Collaboration Protocol Profile (CPP) and Agreement (CPA)** specifications [OAS02]. tpaML and CPP/CPA capture the interoperation parameters (e.g. message formats, communication protocols, etc.), but make no provisions for fulfillment monitoring. Microsoft's **BizTalk** has auditing and an optional document mining function as well [MLA+02]. These frameworks aim to support recovering from failures, instead of prevention. Consequently, the development of contract models for contract fulfillment monitoring is relatively unexplored. Our approach provides pro-active detection of imminent contract violations.

Ludwig et al. propose a Service Level Agreements (SLA) language for dynamic electronic services [LKD+03]. SLA for web services is specified and monitored under the WSLA (Web Service Level Agreement) framework [KL03]. It includes parties, service definitions and obligations. The *parties*

are involved in the management of the web service, The *service definitions* describe the service properties on which obligations are defined. The *obligations* define the service level that is guaranteed with respect to the SLA parameters. Their work takes a Quality of Service (QoS) view on monitoring web services. All services and obligations are explicitly defined and can be measured. This is different from monitoring contractual parties' behavior, as described in this paper. SAL quantities performance of business process. SLA is not a pro-active monitoring requirement. SLA is a specific contract used only for monitoring the QoS of web services, but our monitoring research focuses on the monitoring issues of general contracts. Our approach can be regards as being price to SLA: it can deliver the input to on SLA monitoring system.

This section reviewed the current research on business process automation and compared that research with our own. In short, our research concerns a monitoring issue in business process automation. The next section discusses contract-related frameworks and architectures.

2.4 Contracting frameworks or architectures

The **Business Contract Architecture (BCA)** [MBBR95] [Mil95] consists of *contract repository*, *notary*, *legal rules repository*, *contract validator*, *contract negotiator*, *contract arbitrator*, *contract monitor*, and *contract enforcer*. BCA is a CORBA-based architecture. BCA uses CORBA interface Definition Language (IDL) to specify the different BCA contract types, and the Interface Repository (IR) to represent an elementary type repository.

During the existence of a contractual arrangement, an object may check whether the contractual obligations have been met during the contract realization. The contract monitoring process can be performed by the objects themselves. Alternatively, a trusted third-party Contract Monitor (CM) object can be used to detect contract violation. The contract enforcement can be done via third party objects: Contract Enforcers (CE).

BCA does not provide generic monitoring facilities, expecting each application to develop its own monitoring code to detect and signal nonconformance to the contract monitor; contract enforcement is limited to either violation signaling and/or preventing the non-performing party from entering into further contracts.

In paper [Dao98], Daoud proposed a business contracting model for Telecommunication Information Networking Architecture (TINA). The contract framework includes a validation module, a negotiation module, a monitoring module and an enforcement module. The concept of this contract framework is much the same as that of BCA.

The basic aims of **SeCo** (secure electronic contracts) [GSSS00], [Dao98], [RSSS99], [Dao98] are developing a generic framework for market services

that can support the process of contract negotiation, signing and settlement, and developing a concept for an electronic contracting container that allows the contracting parties to collect and process all the transaction information that is useful to enforce the contract. The contracting services, which are based on the work of BCA, as well as some other market services and logistics services, include validation services, negotiation services, monitoring services, enforcement services, arbitrating services and repository services. The monitoring services allow events to be triggered according to the current state of the contract and alert an enforcement service to initiate an enforcement activity. The enforcement service supports the process of indicating that the contract has not been honored by another party and performs the corrective actions.

The SeCo project presents a Business Media Framework (BMF), and analyzes the contracting services which include concepts of monitoring services, pro-active enforcement and reactive enforcement. A trusted third-party contract monitor can be used in this framework. However, the problem of how to implement them is not addressed.

In paper [GLA02], the authors present a three-level process and data specification framework for dynamic contract-based service outsourcing and discuss an abstract architecture for dynamic service outsourcing based on the three-level framework. It is shown how the framework and architecture can be placed in the context of existing infrastructures for cross-organizational process support. In this paper, the contract is one kind of fixed contract template, which involves only two contractual parties: an external party and an internal party. Comparing with our research perspectives, this framework mainly focus on how to separate the contractual obligations into the external and internal party. Speaking generally, this paper did a vertical level research which is involved with workflow system details. On the other hand, our research is a horizontal level research which is interested in interactions among multiple contractual parties and our concept is platform-independent.

Another important project worthy of mention is the Coyote Project [DDN+98], [DP97a], [DP99], [DP97b]. Coyote's architecture includes a Business Services Application (BSA) manager in which service contracts and business logics are registered independently with run time services, a persistence and recovery component, an error-handling component and a security component.

The Coyote project is an event reaction and rule-based application. In this application structuring approach, rules are supplied to determine which pieces of application logic should be triggered by different combinations of events [DDN+98] [DP99], user interactions, completions, arriving requests, etc. The monitor uses these to control application processing. The way of structuring logic is particularly effective in highly parallel environments in which related actives may complete in different orders and under complex

conditions. In the Coyote project, event reaction monitors have rules that depend on the previous state of the conversation [DDN+98] [DP97a]. For long-running applications, the infrastructure must provide some effective automated scheme for retrieving the conversation state rather than leaving this to the application. A disadvantage of simple event reaction monitors is that as the number of rules becomes large it becomes increasingly difficult to understand the resulting flow through any intended business process. The structure of the service contract is comparable with the structure of ebXML's TPA [DND+01] or OASIS's Collaboration Protocol Agreement (CPA) [OAS02].

Daskalopulu et al. [DDM01], [DTM02], also introduce a trusted third-party to an e-market architecture for contract performance monitoring.

This section discussed contract-related frameworks and architectures. These will be developed in Chapter 6. As we mentioned, most frameworks or architectures use a trusted third-party for contract monitoring.

This chapter reviewed the literature from different dimensions. First, we focused on monitoring-related issues on event-based systems and multi-agent systems, how a contract concept is used in programming languages, and how a contract could be represented from an Artificial Intelligence point of view. Second, we reviewed contract related logics, addressing the relationship between each logic and its application for particular contract issues. Third, we discussed existing contract models and languages, specifying which particular problems are solved by different contract models. At the same time, we introduced the research concern of our contract model. Finally, we discussed contract-related frameworks and architectures.

The subsequent chapters introduce the temporal logic we propose, along with our monitorable contract formal model. We also discuss our monitoring mechanism and a framework for monitorable contract model for our contract fulfillment monitoring.

Chapter 3

Temporal Logic

Logic is a well-established technique to formalize problem areas and to describe solutions by means of calculus. *First-order predicate logic* is a predicate logic in which predicates take only individual arguments and quantifiers only bind individual variables. *Deontic logic* is the study of the logical relationships among propositions that assert that certain actions or states of affairs are morally obligatory, morally permissible, morally right or morally wrong.

Most classic contract models use first-order logic and deontic logic together for task allocation and process co-ordination [WX01] [TT99] [NSJ98] [Lee98c] [WDV97]. Deontic logic is also used for obligation and right models for e-service contracts [LS03]. In legal e-contract systems, deontic logic formalizes the duties, rights and other complex legal concepts [Pro92]. It clarifies and expands an incomplete and imprecise statement of requirements into a precise formal specification.

In standard logic, propositions are considered to be either true or false. However, it is not enough in the contract fulfillment stage when each contractual party has a different view on its own and the counter-parties' behaviors and whether they comply with the agreed contract. Thus a new logic, subjective logic, has been proposed. *Subjective logic* [Jös99] [Jös01] addresses the problem of forming a measurable belief about the truth or falsity of an atomic proposition asserting a property of the world, and use the term *opinion* to denote the representation of a subjective belief. Subjective logic can be seen as an extension of both probability calculus and binary logic [Jös01].

Daskalopulu et al. explored evidence-based contract monitoring [DM01] [DDM01] [DTM02] and contract enforcement [MsDP02] using subjective logic. They deal with disputes between parties. To be able to resolve the disputes, each partner should normally present relevant evidence concerning factual aspects of the transaction to a commonly accepted arbitrator in conventional contracting.

Our research follows a different perspective from the logic contract models and the logic-based contract monitoring mechanisms we mentioned above. Our approach is more realistic by allowing contract violations rather than giving different weights to preferred actions or explicitly marking certain actions as forbidden.

Subjective logic is not suitable for formal contract monitoring because it leaves the decision about whether a violation has taken place to the arbitrator. Moreover our contract model deals with likelihood of propositions rather than clear evidence of a contract violation. Deontic logic is designed to guide the behavior of agents. In multi-partner contracts however, a partner should be considered free in her decision to comply or not comply with her contractual obligations. We are interested in detecting real or imminent violations but would not qualify any action or absence of action to be allowed or forbidden.

In our research, we shall not distinguish permitted or forbidden action but rather reason about whether the relevant action has occurred or can occur in the future. For this we use temporal logic, temporal logic is logic of propositions whose truth and falsity may depend on time [Kro87], it provides a possibility for pro-active monitoring.

Section 3.1 explains why we need a new propositional temporal logic (PTL). Section 3.2 compare our PTL to the mainstream temporal logic. Section 3.3 provides properties of our specification. Section 3.4 explains our PTL and provides a formal syntax and semantics. This chapter ends with a summary in section 3.5.

3.1 Technical motivation

E-contract monitoring would be facilitated by an infrastructure that includes a generic and rigorous approach to e-contract specification and scheduling.

In event-based monitoring system, GEM [MSS97] specifies the operator of events, but does not address scheduling. In transaction models, *intertask dependencies* of the transaction models are formalized by using the temporal logic CTL (Computational Tree Logic) [ASSR93]; workflow dependencies are formalized by using self-defining temporal logic [Sin97]. These approaches may correspond to monitor actions of different contractual parties. We need a new propositional temporal logic which should facilitate to trace contractual actions at run-time.

3.2 Comparison of mainstream and our PTL

Temporal logic can be used to specify processes, and to reason about their properties. The idea of using temporal logic as a specification language, is of course not new. In theoretical computer science, many temporal logic for

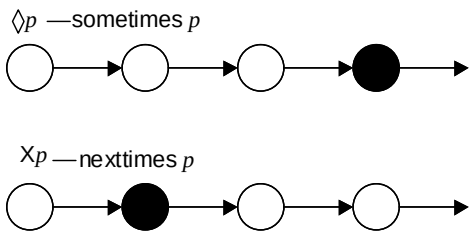


Figure 3.1: Intuitive meaning for linear-time operator [Eme90]

specifying and reasoning about the behavior of processes have been proposed and studied in [Eme90], [MP92]. The behavior specified in these temporal logics, is not reasoning behavior but the behavior of hardware processors in a computer system [Eng99]. However, a state of a processor at a certain point in time is very different from reasoning a contractual party's behaviors, which we call actions. Differences between the mainstream temporal logics and our propositional temporal logic (PTL) are discussed in following sections.

3.2.1 Differences with the standard linear-temporal logic

In [Eme90], semantics of a formula of Propositional Linear Temporal Logic (PLTL) are defined with respect to a linear-time structure $M = (S, x, L)$. The linear-time structure M is formalized as a time-line notion, where

- S is a set of *states*.
- $x : \mathbb{N} \to S$ is an infinite sequence of *states*, and
- $L : S \to \text{PowerSet(AP)}$ is a labeling of each state with the set of atomic propositions in AP true at the state.

The semantics of PLTL thus refer to the *states*. In our research new temporal logic should be introduced and studied as reasoning *actions*. Thus, the first difference is that our temporal logic does refer to the *states*.

Basic temporal operator of propositional linear temporal logic are $\Diamond p$ ("sometime p"; also read as "eventually p") and $\mathsf{X}p$ ("nexttime p"). Figure 3.1 illustrates their intuitive meanings.

Because of the semantics of PLTL with respect to the states, it is natural for a formula of PLTL that it can change from *true* to *false* and reverse following different the states. In our action reasoning, the stability of actions implies that an action once occurred is *true* forever. The actions hence do not refer to any states. It is the second difference with PLTL.

Some temporal logic formalisms have been based on temporal operators that are evaluated as *true* or *false* of *points* in time. Some formalisms, however, have temporal operators that are evaluated over *intervals* of time.

In our application, because the action may have to be pro-actively triggered, or may be allowed to occur immediately, use of intervals greatly simplifies the formulation of certain properties.

In short, there are three differences between our PTL and the main stream PLTL. Our PTL does not refer to the "states"; the action once occurred is "true" forever, some temporal operators have different meanings; and our PTL requires a interval-time structure. In the next section, a comparison between the trace semantics of the labeled transition system and our trace semantics of PTL is discussed.

3.2.2 Differences with trace semantics of labeled transition systems

A class of sequential processes can often be conveniently represented as a labeled transition system. This is a domain $\mathbb{P}$ on which infix written binary predicates $\xrightarrow{a}$ are defined for each action $a \in Act$ where Act is a given set of performing actions in processes. The elements of $\mathbb{P}$ represent processes, and $p \xrightarrow{a} q$ means that p can start performing the action a and after completion of this action reach a state where q is its remaining behavior. In a labeled transition system it may happen that $p \xrightarrow{a} q$ and $p \xrightarrow{b} r$ for different actions a and b or different processes q and r. This phenomenon is called *branching*. We list definitions [vG01] of the *labeled transition system* and *action relationships* as follows:

- a *labeled transition system* is a pair $(\mathbb{P}, \rightarrow)$ with $\mathbb{P}$ a class and $\rightarrow \subseteq \mathbb{P} \times Act \times \mathbb{P}$, such that for $p \in \mathbb{P}$ and $a \in Act$ the class $\{(p, a, q) \in \rightarrow\}$ is a set.

- The *generalized action relations* $\xrightarrow{\sigma}$ for $\sigma \in Act^*$ are defined recursively by:

  1. $p \xrightarrow{\varepsilon} q$, for any process p.
  2. $(p, a, q) \in \rightarrow$ with $a \in Act$ implies $p \xrightarrow{a} q$ with $a \in Act^*$.
  3. $p \xrightarrow{\sigma} q \xrightarrow{\rho} r$ implies $p \xrightarrow{\sigma\rho} r$.

  Where Act^* be the set of finite sequences of Act. In words, the generalized action relations $\xrightarrow{\sigma}$ are the reflexive and transitive closure of the ordinary action relations $\xrightarrow{a}$. $p \xrightarrow{\sigma} p$ means that p can evolve into q, while performing the sequence σ of actions.

- The set of *initial actions* of a process p is defined by: $I(p) = \{a \in Act | \exists q : p \xrightarrow{a} q\}$.

The complete trace semantics are defined as follows:

- $\sigma \in Act^*$ is a *complete trace* of a process p, if there is a process q such that $p \xrightarrow{\sigma} q$ and $Iq = \emptyset$. Let $CT(p)$ denote the set of complete traces of p. Two processes p and q *are completed trace equivalent*, notation $p =_{CT} q$ denote the set of complete traces of p.

- The set $\mathcal{L}_{CT}$ of completed trace formulas over Act is defined recursively by:

  - $\top \in \mathcal{L}_{CT}$.
  - $0 \in \mathcal{L}_{CT}$.
  - If $\varphi \in \mathcal{L}_{CT}$ and $a \in Act$ then $a\varphi \in \mathcal{L}_{CT}$.

- The *satisfaction relation* $\models \subseteq \mathbb{P} \times \mathcal{L}_{CT}$ is defined recursively by:

  - $p \models \top$ for all $p \in \mathbb{P}$.
  - $p \models 0$ if $I(p) = \emptyset$.
  - $p \models a\varphi$ if for some $q \in \mathbb{P} : p \xrightarrow{a} q$ and $q \models \varphi$.

Note that a completed trace formula satisfied by a process p represents either a trace (if it has the from $a_1 a_2 \cdots a_n \top$) or a completed tract (is it has the from $a_1 a_2 \cdots a_n 0$).

Again, the first different with our temporal logic, completed trace semantics refer to the states because this semantics of CTL express the process of a black box. The process autonomously chooses an execution path that is consistent with its position in the labeled transition system $(\mathbb{P}, \rightarrow)$. During this execution always an action name is visible on the display. As soon as no further action can be carried out, the process reaches a state of deadlock and the display becomes empty.

Our action reasoning is about action sequences. We specify which action should occur within certain intervals. After action occurrences, the state of contract performing is out of the current concern. The second difference is that we choose linear-time to specify our new temporal logic, not branching-time temporal logic.

In addition, we specify $\bar{a}$ as a complement of action a ($a \in Act$). The reason why we need $\bar{a}$ is explained using a case. For example, in an internet-based e-market, action *cancel* disables action *shipping* after an order. In other words, if action *shipping* occurs, action *cancel* did not occur before. Using $\bar{a}$ operator, we can write as $\overline{cancel} \cdot shipping$. The third difference with our temporal logic is that the complement of action is not specified at the completed trace definition.

Generally, there are three differences of the trace semantics between CTL and our PTL. Our trace semantics do not refer to the "states", is based on the linear-time, and uses a new alphabet which is facilitated to explain some common situations in the e-contracting processes.

In Sections 3.2.1 and 3.2.2, we compare my requirements for the new temporal logic with the main stream linear temporal logic (PLTL)and computational tree logic (CTL), respectively. Therefore, a new temporal logic should be introduced and studied which is suited in particular to describing the action relationships between contractual parties over time of a reasoning contract fulfillment.

3.3 Properties of our PTL

One obvious choice for the formalization of time, is to use linear time logic. In Linear time logics, time is treated as if each moment in time has a unique possible future. But another possibility is to use branching time structures. In branching time logics, each moment in time may split into various possible futures. In theoretical computer science there has been much debate whether time should be dealt as linear or branching [Pnu85], [Var01]. The most important differences between these two approaches are that the linear time logics have in general an easier expressiveness and a lower complexity, but the branching time logics have an easier verification.

Linear time logic formulas are interpreted over linear sequences and execution of actions are evaluated over intervals of time. In order for a temporal logic to be a specification language for the action relationship between contractual parties, it should be able to describe traces which are sequences of actions. The propositional temporal logic used for this thesis can thus be regarded describing the action relationship between contractual parties as a linear time logic. This logic is mainly based on number of previous work: [Sin97],[CL90], and [Woo92]. This logic (further described in Section 3.4) was described earlier in [XJ03] †.

The logic used is a temporal logic focussed on describing sequences of actions. While not the same, this concept is close in semantics to event based temporal logic, that describes sequences of events. This concept of event temporal logics dates back to at least 1957 when it was used in work by Prior [Pri57]. Event-based temporal logic has been very popular in the field of computer science, especially when addressing multi-agent systems and workflow systems, used for example in [Woo92] and [WJ94], but has become less popular in more recent times†.

Singh [Sin97] presents an example of such an event based logic, to describe sequences of events in the context of agent systems. The semantics of that logic are primarily based upon [CL90] and [Woo92] that in themselves share important semantics, but other well-known temporal semantics have been added as further detailed in Section 3.4.2. The papers main contribution are found in the application in the agent domain, as well as the presentational elegance of its syntax, in particular in its use of a $\cdot$ operator to denote the well-established before semantic. This presentational

elegance and proximity to the requirements of this thesis lead to its selection as the basis of the logic presented below. Differences between both logics are mainly found in their higher level constructs were the differences between events and actions become more important, as well as the differences in context between agent systems on the one hand and electronic contract monitoring on the other[†].

The properties of our proposition temporal logic include:

- linear-time temporal logic over intervals,
- supporting trace semantics, and
- supporting an expression of the complement of an action.

In the next section the syntax and semantics of our PTL are defined.

3.4 Propositional temporal logic (PTL)

Temporal logic has been proposed as a framework with which we may reason about a changing world [AM90]. It enables us to describe phenomena that occur over time. For example, we may write formulas to denote "action a occurs before action b" as $a \cdot b$, and "action a eventually occurs" as $\Diamond a$. The properties of temporal logics satisfy the need of the research in contract pro-active monitoring.

We propose a *propositional temporal logic* (PTL) to formalize some issues with regard to which actions have occurred, which have not yet occurred but are expected to occur, and which will never occur. In the rest of this section we define the syntax and semantics of PTL in sections 3.4.1 and 3.4.2, respectively.

3.4.1 Syntax

A *language* $\mathcal{T}$ of PTL is countable collection of proposition symbols. $\mathcal{A} \neq \varnothing$ is the set of significant actions; $\Sigma = \{a, \overline{a} : a \in \mathcal{A}\}$ is an *alphabet*. a means that action a occurs at certain interval of time; $\bar{a}$ is action a's complement, and $\bar{a}$ denotes that action a does not occur during a certain interval.

In our propositional temporal logic, the symbols $-, \neg, \Diamond, \Box$ and $\cdot$ are introduced as follows:

1. $\overline{a}$ denotes action a does not occur at certain interval of time;
2. $\neg a$ denotes that action a has not yet occurred,
3. $\Diamond a$ denotes that action a will eventually occur,
4. $\Box a$ denotes that action a has occurred; in other words, action a is always true from now on,

5. $a \cdot b$ denotes that action a occurs before action b.

Below we give the syntax as follows:

Syntax 3.4.1.1 *Let $\mathcal{T}$ be the temporal language; $\mathcal{A}$ is the set of actions, $\Sigma = \{x, \bar{x}, : a \in \mathcal{A}\}$ is an alphabet, $x, x_1, x_2 \in \Sigma$; 0 refers to a specification that is always false, $\top$ refers to one that is always true. The syntax of $\mathcal{T}$ is inductively defined by*

1. $\Sigma \subseteq \mathcal{T}$,
2. $0, \top \in \mathcal{T}$,
3. $\neg x \in \mathcal{T}$ *for all* $x \in \Sigma, x \in \mathcal{T}$,
4. $\Diamond x \in \mathcal{T}$ *for all* $x \in \Sigma, x \in \mathcal{T}$,
5. $\Box x \in \mathcal{T}$ *for all* $x \in \Sigma, x \in \mathcal{T}$,
6. $x_1 \vee x_2 \in \mathcal{T}$ *for all* $x_1, x_2 \in \Sigma, x_1, x_2 \in \mathcal{T}$
7. $x_1 \wedge x_2 \in \mathcal{T}$ *for all* $x_1, x_2 \in \Sigma, x_1, x_2 \in \mathcal{T}$,
8. $x_1 \cdot x_2 \in \mathcal{T}$ *for all* $x_1, x_2 \in \Sigma, x_1, x_2 \in \mathcal{T}$,
9. *no more expressions are in* $\mathcal{T}$.

We define the following binding precedence in decreasing order: $\neg; \cdot; \Box; \Diamond; \wedge; \vee$.

3.4.2 Semantics

The semantics of $\mathcal{T}$ characterize the progress along a given trace, precisely what is needed to determine after an action be tried. The semantics of $\mathcal{T}$ are given with respect to a trace and a pair of indices into that trace. For $0 \leq i \leq j$, $t \models_{i,j} a$ means that a is satisfied over the subsequence of t between i and j.

Definition 3.4.2.1 *A trace, t, is a sequence of actions which if and only if for each action, either the action (e.g. action a) or its complement (e.g. $\bar{a}$) occurs on t.*

For example, traces are written as action sequences enclosed in $<$ and $>$ brackets.

Example 3.4.2.2 *Let $t =< a_1 a_2 a_3 \cdots >$ be a trace in $\mathrm{T}_{\mathcal{T}}$. It is easy to verify the following:*

(a) $t \models_{0,0} \Diamond a_3$

(b) $t \not\models_{0,0} \Diamond(a_3 \cdot a_2)$

(c) $t \models_{0,1} \Box a_1 \wedge \neg a_2 \wedge \neg a_3$

(d) $t \not\models_{0,1} (a_1 \cdot a_3)$

(e) $t \models_{0,3} (a_1 \cdot a_3)$. *In this case,* $(a_1 \cdot a_3)$ *is satisfied when it is in the past of the given index.*

For example, $a \in \mathcal{A}$, $\bar{a}$ is the complement of action a. $t \models_{i,j} a$ means action a occurs at trace t between index i and j; $t \models_{i,j} \bar{a}$ means action $\bar{a}$ occurs at trace t between index i and j.

A trace is a sequence of actions and complements of the actions over time. If a denotes an action, then $\bar{a}$ is the complement of action a. $\bar{a}$ is explicitly included in the trace to state that action a was explicitly not performed by a partner. It can be determined by deadlines or by direct inclusion into the trace by the responsible partner.

Semantics 3.4.2.3 *Let* $\mathcal{T}$ *be the temporal language,* $\mathcal{A}$ *is the set of actions,* $\Sigma = \{a, \bar{a}, : a \in \mathcal{A}\}$ *is an alphabet,* $x, x_1, x_2 \in \Sigma$. *The semantics of* $\mathcal{T}$ *are defined as follows:*

1. $t \models_{i,j} x$ *iff* $\exists k : i \leq k \leq j$ *and* $t \models_{0,k} x$
2. $t \models_{i,j} x_1 \vee x_2$ *iff* $t \models_{i,j} x_1$ *or* $t \models_{i,j} x_2$
3. $t \models_{i,j} x_1 \wedge x_2$ *iff* $t \models_{i,j} x_1$ *and* $t \models_{i,j} x_2$
4. $t \models_{i,j} x_1 \cdot x_2$ *iff* $(\exists k : i \leq k \leq j$ *and* $t \models_{i,k} x_1$ *and* $t \models_{k+1,j} x_2)$
5. $t \models_{i,j} \neg x$ *iff* $t \not\models_{i,j} x$
6. $t \models_{i,j} \Diamond x$ *iff* $(\exists k : j \leq k$ *and* $t \models_{i,k} x)$
7. $t \models_{i,j} \Box x$ *iff* $(\forall k : k \leq j \Rightarrow t \models_{i,k} x)$

Rule 1 refers to the immediate past, bounded by the indices of the semantic definition, action a occurs, or action a does not occur. This concept was already used by Kripke in 1965 [Kri65], but can also be found in for example [Woo92], [CL90], [V$^+$90], and [Sin97] †.

Rule 4 refers to the immediate past as well and can be found from eg.[All83], [CL90], [Eme90], [Woo92], [AF94], [WJ94], and[Sin97] †.

Rules 2, 3, and 5 are semantics for conjunction, disjunction and negation as used in conventional non-temporal logic, adapted for the temporal case in a straightforward way, with the expected semantics. Examples of similar definitions to rule 2 can be found in [Kri65], [Woo92], [CL90], and[Sin97]. Examples for rule 3 are found in [Kri65], [V$^+$90], [HS91], and[Sin97]. Finally, examples of rule 5 are found in [Kri65], [Woo92], [CL90], [V$^+$90], [HS91], and[Sin97] †.

Rule 6 looks into the future, defining the semantics of the $\Diamond$ operator. Examples of this semanitic (but not always with similar syntax) can be found in [CL90], [Woo92], [WJ94], and[Sin97] †.

Rule 7 looks into the present and future in terms of the $\Box$ operator. Examples of this semantic can be found in [CL90], [Woo92], [WJ94], and[Sin97]. All symbols as defined by these semantics will be used in following chapters. While there are strong similarities with the syntactical semantics as presented in [Sin97] and the propositional temporal logic used in this thesis, it must be made clear however that both lean heavily on previous work in temporal logic: 7 of the 9 semantics in [Sin97] already occurred in the work by Cohen and Levesque as published in 1990 [CL90]. The other semantics were also used throughout the field of temporal logic, without coming to a well-established commonly used standard. The same holds for the approach to semantics using double indexed intervals, and modal temporal connectives †.

The formula a is *satisfiable* if some standard model t satisfies a; that is, $t \models a$. The formula a is *valid* if all standard models satisfy a; we denote this by $\models a$.

3.5 Summary

We have developed a temporal language that can express the knowledge necessary to execute actions eagerly. This language requires a different formal semantics than previous temporal logics. We have compared our PTL to the previous temporal logics. Finally, the syntax and semantics of PTL were provided.

Chapter 4

A Formal Model of Monitorable Contracts

A contract between multiple business partners contains some statements about their business relationship, in particular on their physical and informational actions. To support business automation between organizations, monitorability of contract execution is crucial [XJ03]. This chapter seeks to explore possibilities for the representation of contracts that can be automatically monitored at the contract fulfillment stage.

As was noted in the previous chapter, PTL will be used to represent constraints in our contract model, which provide rigorous formal semantics for contract computation. Moreover, we show how those contract constraints can be used to reason about contract scheduling.

Section 4.1 provides an overview of the monitorable contract model, which includes a trade process part, a logic relationship part and a commitment graph. Section 4.2 presents the trade process part of our monitorable contract model, which describes what each partner actually needs to do, and specifies some guarantees between contractual partners. Section 4.3 describes logic relationships, which capture how far a business process has progressed. Section 4.4 discusses the commitment graph, which shows an overview of all commitments of a contract. Section 4.5 presents a formal model of our monitorable contract. This chapter ends with a summary in Section 4.6.

4.1 Overview of monitorable contract model

A contract is an agreement between two or more parties that is binding to those parties and that is based on mutual commitments [WX01]. Our monitorable contract model (in Figure 4.1) consists of three core components: the trade process, the logic relationship and the commitment graph [Xu02a]. The *trade process* specifies the actions and commitments involved. An ac-

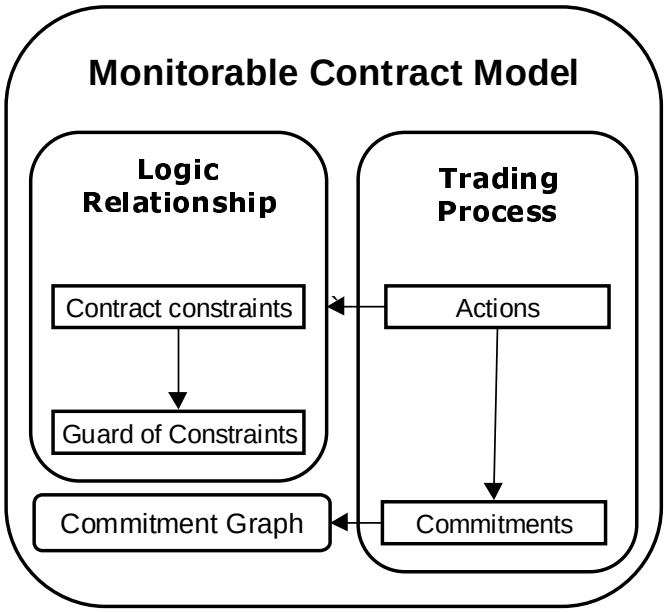


Figure 4.1: The monitorable contract model

tion describes what each partner should do. A commitment in this paper is defined as a guarantee by one party towards another party that some action sequence shall be executed completely provided that some "trigger, involve and finish" action happens, and all involved parties fulfill their side of the transaction. To finish a commitment, there are more than one party involved to finish relevant actions. The *logic relationships* includes contract constraints and guards of contract constraints. The contract constraints describe the logic relationship between actions and show the occurrence order between actions in a business process. The guards of contract constraints specify the right order of actions, checking what obligations remain to be realized after the occurrence of the guarded action. A *commitment graph* is an overview of commitments between partners (i.e., it is the encoding of some contract clauses). The graph is also a visual tool to show commitment relationships in a contract. All of these components of the monitorable contract model will be explained in turn in the next sections.

4.2 Trading process

The trading process is a direct mapping from a paper contract to an electronic contract. In this part of e-contract, we formalize actions and commitments. The actions indicate what each partner should do, whereas the commitments specify a guarantee between contractual partners.

Based on the car insurance case in Chapter 1, actions and commitments

of the parties involved (AGFIL, Lee C.S., Garage, Euro Assist and policyholders) are identified and specified in Section 4.2.1 and Section 4.2.2, respectively.

4.2.1 Actions

An action is an atom in our contract model. An action is specified by an action name, a sender of the action, a receiver of the action and a deadline before which the action must be performed. Because a contract party can be involved in different commitments and play the different roles, we specify the roles of a party as $\mathcal{R}$. A set of total roles of a contract is denoted as $\mathbb{R}$.

Definition 4.2.1.1 *A party can act under different roles in different commitments. Let ID be a domain of identifier;* **roles of a party** *$\mathcal{R}_x$ is defined as*

$$\mathcal{R}_x \subseteq ID.$$

Let $\mathcal{P}$ be a set of parties, the **set of all roles** *is*

$$\mathbb{R} = \bigcup_{\forall x \in \mathcal{P}} \mathcal{R}_x$$

Example 4.2.1.2 *In car insurance, the following abbreviations was used for different contractual parties: P for Policyholders; AG for AGFIL; E for Europ Assist; L for Lee Consulting Services; G for Garage; and A for Assessor. All parties are included in a set $\mathcal{P}$:*

$$\mathcal{P} = \{P, AG, E, L, G, A\}.$$

The garage (G) involved in the repair service *commitment, acts as a repairer: role G'. In the* daily service *commitment, the garage is a cooperator with Lee C.S.: role G''. In the* pay repair cost *commitment, the garage acts as a receiver: role G'''. Hence, the garage plays three roles: G', G'', G'''. All of roles enacted by the garage can be expressed as a set $\mathcal{R}_{garage}$:*

$$\mathcal{R}_{garage} = \{G', G'', G'''\}.$$

The policyholder (P) is involved in the phone service *and* claim form *commitments. In the* phone service *commitment, the policyholder acts as a client of AGFIL. In the* claim form *commitment, the policyholder is a responder to return the claim form. All the roles enacted by the policyholder can be expressed as a set $\mathcal{R}_{policyholder}$:*

$$\mathcal{R}_{policyholder} = \{P', P''\}.$$

In the car insurance case, the set of all roles $\mathbb{R}$ can be specified as:

$$\mathbb{R} = \{P', P'', E, AG, L, A, G', G'', G'''\}.$$

An action has a particular *name* from domain ID, a *sender* of the action and a *receiver* of the action from $\mathbb{R}$ and a performance *deadline* of the action from time $\mathbb{T}$. Deadlines provide the flexibility to initiate actions at a more optimal time, rather than immediately invoking operations the moment a condition occurs.

Definition 4.2.1.3 *Let $\mathbb{R}$ be a set of all roles of all parties, ID be the domain ID, and $\mathcal{T}$ be the time. An* **action** *is specified as*

$$action = (name, sender, receiver, t)$$

where $name \in ID, sender, receiver \in \mathbb{R}$ and $t \in \mathbb{T}$. We require all names of actions to be unique so they can be used as identifiers.

A set of actions $\mathbb{A}$ for a contract can be specified as

$$\mathbb{A} = \bigcup_{\forall x \in \mathcal{P}} \{action\}.$$

For car insurance case, all actions are specified in the following tables.

Example 4.2.1.4 *The prefix "A_" indicates an action name. The second argument denotes the sender of the action; the third argument denotes the receiver of the action; and the last parameter is the deadline of the action. We classify actions based on the sender of the action; the specification and description of actions are summarized in following tables.*

Actions of Policyholder	
(A_phoneClaim,P',E,0)	The policyholder phones Europ Assist to report the car damage.
(A_receiveInfo,P',E,0)	The policyholder send information to Europ Assist.
(A_returnClaimForm,P'',AG,10)	The policyholder needs to return the claim form to AGFIL during reporting the car damage 10 days.
(A_sendCar,P',G',1)	The policyholder must send the car to the garage during reporting the car damage claim 1 day.

According to contracts between partners, each action should support certain commitments. The next section specifies commitments that are the key part of contracts.

4.2.2 Commitments

In this paper, a commitment is a guarantee by one party towards another party that some action sequence shall be executed completely provided that

Actions of AGFIL	
(A_sendClaimForm,AG,P",0.6)	AGFIL need to send the claim form to the policyholder during AGFIL knows the car damage claim 0.6 days.
(A_forwardClaim,AG,L,1.0)	AGFIL need to forward the claim to Lee Consulting Services during the car damage calim receiving 1 day.
(A_payRepairCost,AG,G"',30)	AGFIL pays the repair cost to the garage during the car damage claim received 30 days.

Actions of Europ Assist	
(A_assignGarage,E,P',0)	Europ Assist immediately assigns a garage for the policyholder.
(A_notifyClaim,E,AG,0.5)	Europ Assist immediately notifies the claim to AGFIL.

Actions of Lee Consulting Services	
(A_contactGarage,L,G",1.5)	Lee Consulting Services contacts the garage during the car damage claim received 1.5 days .
(A_assignAssessor,L,A,1.7)	Lee Consulting Services assigns an assessor to inspect the car if the repair cost more than $500 during the car damage received 1.7 day.
(A_agreeRepairCar,L,G",3.5)	Lee Consulting Services agrees the garage to repair the car during the car damage claim received 3.5 days.
(A_forwardInvoices,L,AG,5)	Lee Consulting Services forwards invoices to AGFIL during the car damage claim received 6 days.

Actions of Assessor	
(A_inspectCar,A,L,3)	The assessor inspects the car for Lee Consulting Services during the car damage claim received 3 days.
(A_sendNewRepairCost,A,L,3)	The assessor sends a new repair cost to Lee Consulting Services during the car damage claim received 3 days.

some "trigger, involve, and finish" action happens, and all involved parties fulfill their side of the transaction. To finish a commitment, there are more than one party involved to finish relevant actions. The notion of commitment in this paper is not related to beliefs, desires, or intentions [KC00] [CL95]. In Cohen and Levesque's research, commitments are related to establishing common beliefs about a certain state of the world [CL95]. In our monitoring

Actions of Garage	
(A_estimateRepairCost,G',P',1.5)	The garage estimates the repair cost for AGFIL during the car damage claim received 1.5 days.
(A_sendRepairCost,G'',L,1.6)	The garage sends the repair cost to Lee Consulting Services during the car damage claim received 1.6 days.
(A_repairCar,G',P',4)	The garage repairs the car for the policyholder during the car damage claim received 4 days.
(A_sendInvoices,G'',L,10)	The garage sends invoices to Lee Consulting Services during the car damage claim received 10 days.

model, we do not reason about beliefs of the contractual parties involved, which Daskalopulu did in evidence-based contract performance monitoring research [DTM02]. We also do not assess the of legal status and directives in business process automation [Abr02a].

For finishing a commitment, in a multi-party contract the actions of different parties are involved. In other words, a contract includes one or more commitments, a commitment includes some actions which could be preformed by multi-partners. Those actions can trigger, involve, and finish the commitment. For example, in the car repair service commitment, the garage first needs to receive the policyholder's car as a trigger of this commitment. Then, the garage will repair the car after Lee C.S. agreed the repair cost, which is estimated by the garage. Actions included in a commitment have different attributes, which we specify as trigger, involve and finish. A commitment is described by a commitment name, sender of the commitment, receiver of the commitment, and a series of actions and their attributes.

Definition 4.2.2.1 *Let $\mathbb{A}$ be a set of actions. For each action $a \in \mathbb{A}$ can trigger(tr), involve(in) or finish (fi) a commitment; hence, each action has its attribute.* **Attributes** *of actions $\mathcal{U}$ can be specified as*

$$\mathcal{U} = \{tr, in, fi\}.$$

Let ID be the domain ID, $\mathbb{A}$ be a set of actions, $\mathcal{P}$ be a set of parties. A **commitment** *is specified as*

$$commitment = (name, sender, receiver, \{(a_1, u_1), (a_2, u_2), \ldots, (a_n, u_n) : a_i \in \mathbb{A}, u_i \in \mathcal{U}\}).$$

where name is an identifer, $name \in ID$; sender and receiver are the contract parties, $sender, receiver \in \mathcal{P}$; $a_1, a_2, \ldots, a_n$ denotes all actions involved in the commitment and their attributes $u_1, u_2, \ldots, u_n$. We require all names of commitments to be unique so that they can be used as identifiers.

A set of commitments $\mathbb{M}$ *can be specified as*

$$\mathbb{M} = \bigcup_{\forall x \in \mathcal{P}} \{commitment\}.$$

Let $N = \{1, 2, 3, \ldots\}$, $a_i \in \mathbb{A}$ *and* $m \in \mathcal{P}$, *a sequence function* $f_{position} : \mathbb{A} \times \mathbb{M} \rightarrow N$,

$$f_{position}(a_i, m) = \begin{cases} i & \text{iff } i \text{ is the sequence number} \\ & \quad \text{of action } a_i \text{ in commitment } m. \\ undef & \text{otherwise.} \end{cases}$$

$f_{position}(a_i, m)$ *denotes the position of action* a_i *in the commitment* m.

The following example shows the commitments of the car insurance case.

Example 4.2.2.2 *The prefix "*C_*" indicates a commitment name. The second capital argument denotes the sender of the commitment; the third capital argument denotes the receiver of the commitment; the following pair sets are all actions involved in the commitment and their attributes.*

We require all names of commitments to be unique so that they can be used as identifiers. In following commitment specification, we use actions' names to identify each action. Every commitment is described as follows:

- C_phoneService*: Europ Assist will offer a phone service to the policyholder. After receiving a phone claim from the policyholder (action* A_phoneClaim *has a trigger attribute), together with all pertinent information (action* A_receiveInfo *has an involve attribute), Europ Assist will assign a garage for the policyholder (action* A_assignGarage *has a finish attribute) and notify AGFIL regarding the policyholder's claim (action* A_notifyClaim *has a finish attribute). Commitment* C_phoneService *is specified as*

$$(\mathsf{C\_phoneService}, \mathsf{E}, \mathsf{P}, \{(\mathsf{A\_phoneClaim}, tr), (\mathsf{A\_receiveInfo}, in), (\mathsf{A\_assignGarage}, fi), (\mathsf{A\_notifyClaim}, fi)\})$$

- C_claimForm*: after AGFIL is informed about the policyholder's claim (action* A_notifyClaim *has a trigger attribute), AGFIL will send the claim form to the policyholder (action* A_sendClaimForm *has an involve attribute); this commitment will be completed after the policyholder returns the claim form to AGFIL (action* A_returnClaimForm *has a finish attribute). Commitment* C_claimForm *is specified as*

$$(\mathsf{C\_claimForm}, \mathsf{AG}, \mathsf{P}, \{(\mathsf{A\_notifyClaim}, tr), (\mathsf{A\_sendClaimForm}, in), (\mathsf{A\_returnClaimForm}, fi)\})$$

- C_dailyService*: Lee C.S. will finish the daily service for AGFIL. After receiving a claim (action* A_forwardClaim *has a trigger attribute), Lee C.S. contacts*

the garage (action A_contactGarage *has an involve attribute), and gets a repair cost from the garage (action* A_sendRepairCost *has an involve attribute). When the estimating repair cost is more than $500, Lee C.S. will assign an assessor to inspect (action* A_assignAssessor *has an involve attribute) and get a new repair cost from the assessor (action* A_sendNewRepairCost *has a trigger attribute). Lee C.S. will send the agreeing repair message to the garage (action* A_agreeRepairCar *has a finish attribute), and the garage repairs the car (action* A_repairCar *has a trigger attribute) and forwards the invoice to AGFIL (action* A_sendInvoices *has an involve attribute). Finally, AGFIL pays the repair cost to the garage (action* A_forwardInvoices *has a finish attribute). Commitment* C_dailyService *is specified as*

$$\begin{aligned}(\mathsf{C\_dailyService}, \mathsf{L}, \mathsf{AG}, \{&(\mathsf{A\_forwardClaim}, tr), (\mathsf{A\_contactGarage}, in),\\ &(\mathsf{A\_sendRepairCost}, in), (\mathsf{A\_assignAssessor}, in),\\ &(\mathsf{A\_sendNewRepairCost}, tr), (\mathsf{A\_agreeRepairCar}, fi),\\ &(\mathsf{A\_repairCar}, tr), (\mathsf{A\_sendInvoices}, in),\\ &((\mathsf{A\_forwardInvoices}, \mathsf{L}, \mathsf{AG}, 5), fi)\})\end{aligned}$$

- C_repairService*: the garage will offer the repair service to the policyholder. After the policyholder sends his/her car to the garage (action* A_sendCar *has a trigger attribute), the garage estimates the repair cost (action* A_estimateRepairCost *has a finish attribute). After the garage receives an agreement from Lee C.S. about the repair cost (action* A_agreeRepairCar *has a trigger attribute), the garage repairs the car (action* A_repairCar *has a finish attribute). Commitment* C_repairService *is specified as*

$$\begin{aligned}(\mathsf{C\_repairService}, \mathsf{G}, \mathsf{P}, \{&(\mathsf{A\_sendCar}, tr), (\mathsf{A\_estimateRepairCost}, fi),\\ &(\mathsf{A\_agreeRepairCar}, tr), (\mathsf{A\_repairCar}, fi)\})\end{aligned}$$

- C_inspectCar*: the assessor reviews the repair cost for Lee C.S. After receiving the requirement from Lee C.S. (action* A_assignAssessor *has a trigger attribute), the assessor inspects the car (action* A_inspectCar *has an involve attribute) and sends the new repair cost to Lee C.S. (action* A_sendNewRepairCost *has a finish attribute). Commitment* C_inspectCar *is specified as:*

$$\begin{aligned}(\mathsf{C\_inspectCar}, \mathsf{A}, \mathsf{L}, \{&(\mathsf{A\_assignAssessor}, tr), (\mathsf{A\_inspectCar}, in),\\ &(\mathsf{A\_sendNewRepairCost}, fi)\})\end{aligned}$$

- C_payRepairCost*: AGFIL will pay the repair cost to the garage for repairing the policyholder's car. After receiving the invoice from Lee C.S. (action* A_forwardInvoices *has a trigger attribute) and a claim form from the policyholder (action* A_returnClaimForm *has a trigger attribute), AGFIL reimburses the repair cost to the garage(action* A_payRepairCost *has a finish attribute). Commitment* C_payRepairCost *is specified as*

$$\begin{aligned}(\mathsf{C\_payRepairCost}, \mathsf{AG}, \mathsf{G}, \{&(\mathsf{A\_forwardInvoices}, tr), (\mathsf{A\_returnClaimForm}, tr),\\ &(\mathsf{A\_payRepairCost}, fi)\})\end{aligned}$$

the set of commitments $\mathbb{M}$ *in the car insurance case is*

$$\mathbb{M} = \{\mathsf{C\_phoneService}, \mathsf{C\_claimForm}, \mathsf{C\_dailyService}, \mathsf{C\_repairService}, \mathsf{C\_inspectCar}, \mathsf{C\_payRepairCost}\}$$

In Table 4.1, we summarize all commitments, actions and action codes which used in the car insurance case.

Commitment	Classification of Actions and Commitments			Codes
	Trigger	Involve	Finish	
C_phoneService (PS)	A_phoneClaim			PS.1
		A_receiveInfo		PS.2
			A_assignGarage	PS.3
			A_notifyClaim	PS.4, CF.1
C_repairService (RS)	A_sendCar			RS.1
			A_estimateRepairCost	RS.2
	A_agreeRepairCar			RS.3, DS.6
			A_repairCar	RS.4, DS.7
C_claimForm (CF)	A_notifyClaim			CF.1, PS.4
		A_sendClaimForm		CF.2
			A_returnClaimForm	CF.3, PR.2
C_dailyService (DS)	A_forwardClaim			DS.1
		A_contactGarage		DS.2
		A_sendRepairCost		DS.3
		A_assignAssessor		DS.4, IC.1
	A_sendNewRepairCost			DS.5, IC.3
			A_agreeRepairCar	DS.6, RS.3
	A_repairCar			DS.7, RS.4
		A_sendInvoices		DS.8
			A_forwardInvoices	DS.9, PR.1
C_inspectCar (IC)	A_assignAssessor			IC.1,DS.4
		A_inspectCar		IC.2
			A_sendNewRepairCost	IC.3,DS.5
C_payRepairCost (PR)	A_forwardInvoices			PR.1, DS.9
	A_returnClaimForm			PR.2, CF.3
			A_payRepairCost	PR.3

Table 4.1: Commitments, actions and action abbreviations

This section specifies actions and commitments that we regard as a direct mapping from a paper contract to an e-contract. Information in the trading process part of our monitorable contract model can compare with contents between "$< action >$" and "$< /action >$" in TAP/CPA of ebXML. The difference between ebXML's action part and our trade process part is that we specify a multi-partner business process using commitments concept and TPA/CPA of ebXML only specified two-partner contract. In the next section, logic relationships between actions are calculated.

4.3 Logic relationship

Once we have specified the actions, we next need to formalize the logic relationships between actions. We use PTL, which is provided in Chapter 3 for formalizing this kind of logic relationship, which we call contract constraints in Section 4.3.1. Furthermore, we depict calculated guards of contract constraints in Section 4.3.2.

4.3.1 Contract constraints

As we observed, there exists logic relationships among actions, which we call contract constraints. Contract constraints express restrictions on the occurrence order among actions in a business process ([Xu03b] and [XJ03]). The formal specification of contract constraints is described as follows:

Constraint 4.3.1.1 *Using Propositional Temporal Logic (in Chapter 3), let* $\mathbb{A}$ *be a set of actions,* Σ *be an alphabet,* $a, b, c \in \mathbb{A}$ *and* $\{a, \overline{a}, b, \overline{b} \in \Sigma\}$. $\overline{a}, \overline{b}$ *or* $\overline{c}$ *denotes* a, b, *or* c *does not occur at certain interval of time respectively. A contract constraint is defined using the following schemes:*

$$\overline{a} \wedge \overline{b} \vee a \cdot b \qquad \text{(scheme 1)}$$

$$((\overline{a} \vee \overline{b}) \wedge \overline{c}) \vee a \cdot b \cdot c \vee b \cdot a \cdot c \qquad \text{(scheme 2)}$$

$$a \cdot b \vee \overline{b} \qquad \text{(scheme 3)}$$

$$a \cdot b \cdot c \vee b \cdot a \cdot c \vee \overline{c} \qquad \text{(scheme 4)}$$

Constraint scheme 1 denotes an *initiation* relationship: a and b can never occur, or a and b occur in order, which means a must occur before b. *Constraint scheme* 2 captures a *joint initiation* relationship: if a or b never occur, c will certainly never occur; if c occurs, a and b both have occurred before. *Constraint scheme* 3 describes an *enable* relationship: if b occurs, that means a has occurred before; otherwise, b will never occur. *Constraint scheme* 4 indicates a *joint enable* relationship: if c occurs, a and b have occurred before – no matter what the order of a and b's occurrence; otherwise, c will never occur. The contract constraint schemes can be extended in different business applications.

The constraint schemes used in this paper have been significantly changed from those used in paper [Sin97]. Only one scheme (scheme 1) is common, but was desribed earlier in numerous works such as [Kle91] and [DKRR98]. Scheme 2, is less straightforward and different in small but important ways. Schemes 3 and 4 are entirely novell, and focussed on the specific requirements of the multi-party electronic contract montitoring contexts†.

It should be noted that the set of schemes for contract constraints can be extended beyond the four schemes presented here. In such case, the guard computation presented subsequently needs to be extended as well.

Example 4.3.1.2 *Based on the car insurance case, the following examples exist for each contract constraint scheme:*

1. *A relationship between action* A_notifyClaim *and action* A_sendClaimForm *belongs to constraint scheme 1: Europ Assist notifies AGFIL of the claim, which initiates AGFIL sending a claim form to the policyholder.*

2. *A relationship among the action* A_forwardInvoices*, action* A_returnClaimForm*, and action* A_payRepairCost *belongs to constraint scheme 2: after receiving the repair invoice from the Lee C. S. and the claim form from the policyholder, AGFIL will pay the repair costs to the garage.*

3. *A relationship between action* A_sendRepairCost *and action* A_agreeRepairCar *belongs to constraint scheme 3: after receiving the repair costs (if less than $500), Lee C. S. can authorize the garage to repair the car immediately.*

4. *A relationship among action* A_estimateRepairCost*, action* A_contactGarage *and* A_sendRepairCost *belongs to constraint 4: after the garage has estimated the repair costs and Lee C.S. has contacted the garage, the garage can send an invoice for repair costs to Lee C.S.*

The section, based on the car insurance case, specifies four kinds of contract constraints. However, many more contract constraints exist in different contracts. For example, in an internet-based e-market, action *cancel* disables action *shipping*. The disable relationship can be described as $cancel \vee \overline{shipping} \vee \overline{cancel} \cdot shipping$. The meaning of the *disable* relationship is that in the certain time before, action *cancel* and action $\overline{shipping}$ can always occur. If action *shipping* has occurred, then action *cancel* never took place before. Contract constraints can be extended case by case. The following section discusses how to use contract constraints to calculate/reason the state of a contract execution.

4.3.2 Guards of contract constraints

For each contract constraint scheme, the guard captures how far that scheme has progressed. After the occurrence of the action, the guard checks what obligations remain to be realized. The guard of an action for a contract constraint refers to which actions have occurred, which have not occurred but are expected to occur, and which should not occur in the future.

The power of the use of temporal logic comes through the use of guards. This is particularly true in the application of temporal logic in the context of computer science and information systems. As such, guards can be found to be defined in many papers. Definitions of such guards can be found in for example [Eme90], [Woo92], as well as [Sin97] that formed the most direct basis for the *syntax* of the used temporal logic. These guards have however been completely reformulated to take into account the different semantics and constraint schemes used[†].

Definition 4.3.2.1 *Let $\mathcal{T}$ be a temporal logic language (see Chapter 3), $\mathcal{W}$ be a set of contract constraints, $\mathcal{S}$ be a constraint scheme in $\mathcal{W}$, $\mathcal{S} \in \mathcal{W}$, and $\mathbb{A}$ be a set of actions. $I_1, I_2 \in \mathcal{S}$, $a_1, \ldots, a_n \in \mathbb{A}$, $\Sigma = \{a, \overline{a} : a \in \mathbb{A}\}$ is an alphabet, $\Sigma \subseteq \mathcal{T}$. A function G: $\mathcal{S} \times \Sigma \mapsto \mathcal{T}$ is the guard of contract constraints. It is inductively defined as*

$$\mathsf{G}(I_1 \vee I_2, a) \equiv \mathsf{G}(I_1, a) \vee \mathsf{G}(I_2, a) \tag{4.1}$$

$$\mathsf{G}(I_1 \wedge I_2, a) \equiv \mathsf{G}(I_1, a) \wedge \mathsf{G}(I_2, a) \tag{4.2}$$

$$\mathsf{G}(a_1 \cdot \ldots \cdot a_n, a_i) \equiv \begin{cases} \top & (i = 1 \ \ and \ \ n = 1) \\ \Box a_1 \wedge \ldots \wedge \Box a_{i-1} \wedge (\neg a_{i+1} \wedge \ldots \wedge \\ \qquad \neg a_n \wedge \Diamond(a_{i+1} \cdot \ldots \cdot a_n)) & (1 \leq i < n) \\ \Diamond(a_1 \cdot \ldots \cdot a_n) & (i < 1 \ \ or \ \ i > n) \end{cases} \tag{4.3}$$

$$\mathsf{G}(\overline{a_1} \cdot \ldots \cdot \overline{a_n}, a_i) \equiv \begin{cases} 0 & (1 \leq i \leq n) \\ \Diamond(\overline{a_1} \cdot \ldots \cdot \overline{a_n}) & (i < 1 \ \ or \ \ i > n) \end{cases} \tag{4.4}$$

$$\mathsf{G}(a_1 \cdot \ldots \cdot a_n, \overline{a}_i) \equiv \begin{cases} 0 & (1 \leq i \leq n) \\ \Diamond(a_1 \cdot \ldots \cdot a_n) & (i < 1, \ \ or \ \ i > n) \end{cases} \tag{4.5}$$

$$\mathsf{G}(\overline{a_1} \cdot \ldots \cdot \overline{a_n}, \overline{a}_i) \equiv \begin{cases} \top & (i = 1 \ \ and \ \ n = 1) \\ \Box\overline{a_1} \wedge \ldots \wedge \Box\overline{a_{i-1}} \wedge (\neg\overline{a_{i+1}} \wedge \ldots \wedge \\ \qquad \neg\overline{a_n} \wedge \Diamond(\overline{a_{i+1}} \cdot \ldots \cdot \overline{a_n})) & (1 \leq i < n) \\ \Diamond(\overline{a_1} \cdot \ldots \cdot \overline{a_n}) & (i < 1, \ \ or \ \ i > n) \end{cases} \tag{4.6}$$

$$\mathsf{G}(0, a) \equiv 0 \tag{4.7}$$

$$\mathsf{G}(\top, a) \equiv \top \tag{4.8}$$

We prove and explain the meaning of guards as follows [Xu03b],

Theorem 4.3.2.2 *Let $\mathbb{A}$ be a set of actions, $a, b \in \mathbb{A}$; Σ be an alphabet, $a, \overline{a}, b, \overline{b} \in \Sigma$, $\mathcal{W}$ be a set of contract constraints, $(\overline{a} \wedge \overline{b} \vee a \cdot b)$ be a contract constraint scheme, $(\overline{a} \wedge \overline{b} \vee a \cdot b) \in \mathcal{W}$. Guards of this contract constraint scheme are*

$$\mathsf{G}(\overline{a} \wedge \overline{b} \vee a \cdot b, a) \equiv \neg b \wedge \Diamond b$$
$$\mathsf{G}(\overline{a} \wedge \overline{b} \vee a \cdot b, \overline{a}) \equiv \Diamond\overline{b}$$
$$\mathsf{G}(\overline{a} \wedge \overline{b} \vee a \cdot b, b) \equiv \Box a$$
$$\mathsf{G}(\overline{a} \wedge \overline{b} \vee a \cdot b, \overline{b}) \equiv \Diamond\overline{a}$$

Proof: we prove these four equations respectively.
According to the **Definition 4.3.2.1 (4.4) and (4.3)**, we have $\mathsf{G}(\overline{a}, a) \equiv 0$, $\mathsf{G}(\overline{b}, a) \equiv \Diamond\overline{b}$ and $\mathsf{G}(a \cdot b, a) \equiv \neg b \wedge \Diamond b$, respectively; according to **Definition 4.3.2.1 (4.2) and (4.1)**, we have $\mathsf{G}(\overline{a}, a) \wedge \mathsf{G}(\overline{b}, a) \equiv \mathsf{G}(\overline{a} \wedge \overline{b}, a)$ and $\mathsf{G}(\overline{a} \wedge \overline{b}, a) \vee \mathsf{G}(a \cdot b, a) \equiv \mathsf{G}(\overline{a} \wedge \overline{b} \vee a \cdot b, a)$. Putting them together yields proof trees

[Pau87] such as

$$\dfrac{\dfrac{\dfrac{\mathsf{G}(\overline{a},a)\equiv 0 \qquad \mathsf{G}(\overline{b},a)\equiv \Diamond\overline{b}}{\mathsf{G}(\overline{a},a)\wedge\mathsf{G}(\overline{b},a)\equiv 0}}{\mathsf{G}(\overline{a}\wedge\overline{b},a)\equiv 0} \qquad \mathsf{G}(a\cdot b,a)\equiv \neg b\wedge\Diamond b}{\mathsf{G}(\overline{a}\wedge\overline{b}\vee a\cdot b,a)\equiv \neg b\wedge\Diamond b}$$

This theorem can be also proved by using semantics definition in Chapter 3.

Because of **Definition 4.3.2.1** *(page 49)*,

$(\bar{a}\wedge\bar{b}\vee a\cdot b),t\models_{i,j} a,$ where $i\le j$ $\iff \bar{a},t\models_{i,j} a,$ where $i\le j$ and
$\bar{b},t\models_{i,j} a,$ where $i\le j$ or
$a\cdot b,t\models_{i,j} a,$ where $i\le j$.

Because of **Definition 3.4.2.1** *(page 36)*,

$\bar{a},t\models_{i,j} a,$ where $i\le j$ $\iff 0.$

$\bar{b},t\models_{i,j} a,$ where $i\le j$ $\iff (\exists k: j\le k$ and $t\models_{j+1,k}\bar{b})$
$\iff t\models_{i,j}\diamond\bar{b}$

$a\cdot b,t\models_{i,j} a,$ where $i\le j$ $\iff (\exists k: j\le k$ and $t\not\models_{i,j} b$ and
$t\models_{j+1,k} b)$
$\iff t\models_{i,j}\neg b$ and $t\models_{j+1,k}\diamond b$
$\iff t\models_{i,j}\neg b$ and $t\models_{i,j}\diamond b$

Thus,

$(\bar{a}\wedge\bar{b}\vee a\cdot b),t\models_{i,j} a,$ where $i\le j$ $\iff t\models_{i,j}\neg b$ and $t\models_{i,j}\diamond b$

According to **Definition 4.3.2.1 (4.6) and (4.5)**, we have $\mathsf{G}(\overline{a},\overline{a})\equiv\top$, $\mathsf{G}(\overline{b},\overline{a})\equiv\Diamond\overline{b}$ and $\mathsf{G}(a\cdot b,\overline{a})\equiv 0$, respectively; according to **Definition 4.3.2.1 (4.2) and (4.1)**, we have $\mathsf{G}(\overline{a},\overline{a})\wedge\mathsf{G}(\overline{b},\overline{a})\equiv\mathsf{G}(\overline{a}\wedge\overline{b},\overline{a})$ and $\mathsf{G}(\overline{a}\wedge\overline{b},\overline{a})\vee\mathsf{G}(a\cdot b,\overline{a})\equiv\mathsf{G}(\overline{a}\wedge\overline{b}\vee a\cdot b,\overline{a})$. Putting them together yields proof trees such as

$$\dfrac{\dfrac{\dfrac{\mathsf{G}(\overline{a},\overline{a})\equiv\top \qquad \mathsf{G}(\overline{b},\overline{a})\equiv\Diamond\overline{b}}{\mathsf{G}(\overline{a},\overline{a})\wedge\mathsf{G}(\overline{b},\overline{a})\equiv\Diamond\overline{b}}}{\mathsf{G}(\overline{a}\wedge\overline{b},\overline{a})\equiv\Diamond\overline{b}} \qquad \mathsf{G}(a\cdot b,\overline{a})\equiv 0}{\mathsf{G}(\overline{a}\wedge\overline{b}\vee a\cdot b,\overline{a})\equiv\Diamond\overline{b}}$$

According to **Definition 4.3.2.1 (4.4) and (4.3)**, we have $\mathsf{G}(\overline{a},b)\equiv\Diamond\overline{a}$, $\mathsf{G}(\overline{b},b)\equiv 0$ and $\mathsf{G}(a\cdot b,b)\equiv\Box a$ respectively; according to **Definition 4.3.2.1 (4.2) and (4.1)**, we have $\mathsf{G}(\overline{a},b)\wedge\mathsf{G}(\overline{b},b)\equiv\mathsf{G}(\overline{a}\wedge\overline{b},b)$ and $\mathsf{G}(\overline{a}\wedge$

$\overline{b}, b) \vee \mathsf{G}(a \cdot b, b) \equiv \mathsf{G}(\overline{a} \wedge \overline{b} \vee a \cdot b, b)$. Putting them together yields proof trees such as

$$\dfrac{\dfrac{\dfrac{\mathsf{G}(\overline{a}, b) \equiv \Diamond \overline{a} \qquad \mathsf{G}(\overline{b}, b) \equiv 0}{\mathsf{G}(\overline{a}, b) \wedge \mathsf{G}(\overline{b}, b) \equiv 0}}{\mathsf{G}(\overline{a} \wedge \overline{b}, b) \equiv 0} \qquad \mathsf{G}(a \cdot b, b) \equiv \Box a}{\mathsf{G}(\overline{a} \wedge \overline{b} \vee a \cdot b, b) \equiv \Box a}$$

According to **Definition 4.3.2.1 (4.6) and (4.5)**, we have $\mathsf{G}(\overline{a}, \overline{b}) \equiv \Diamond \overline{a}$, $\mathsf{G}(\overline{b}, \overline{b}) \equiv \top$ and $\mathsf{G}(a \cdot b, \overline{b}) \equiv 0$ respectively; according to **Definition 4.3.2.1 (4.2) and (4.1)**, we have $\mathsf{G}(\overline{a}, \overline{b}) \wedge \mathsf{G}(\overline{b}, \overline{b}) \equiv \mathsf{G}(\overline{a} \wedge \overline{b}, \overline{b})$ and $\mathsf{G}(\overline{a} \wedge \overline{b}, \overline{b}) \vee \mathsf{G}(a \cdot b, \overline{b}) \equiv \mathsf{G}(\overline{a} \wedge \overline{b} \vee a \cdot b, \overline{b})$. Putting them together yields proof trees such as

$$\dfrac{\dfrac{\dfrac{\mathsf{G}(\overline{a}, \overline{b}) \equiv \Diamond \overline{a} \qquad \mathsf{G}(\overline{b}, \overline{b}) \equiv \top}{\mathsf{G}(\overline{a}, \overline{b}) \wedge \mathsf{G}(\overline{b}, \overline{b}) \equiv \Diamond \overline{a}}}{\mathsf{G}(\overline{a} \wedge \overline{b}, \overline{b}) \equiv \Diamond \overline{a}} \qquad \mathsf{G}(a \cdot b, \overline{b}) \equiv 0}{\mathsf{G}(\overline{a} \wedge \overline{b} \vee a \cdot b, \overline{b}) \equiv \Diamond \overline{a}}$$

□

The guards of contract constraint scheme 1 mean: action a can occur when action b has not yet occurred and may yet occur in the future; action a can never occur and action b can never occur; action b can occur only if action a has occurred; action b will never occur when action a may never occur either.

We explain the use of guards with the running example. When Europ Assist has notified AGFIL about the claim, AGFIL has not yet sent a claim form to the policyholder, but AGFIL may send a claim form soon. If Europ Assist never sends any claim to AGFIL, AGFIL will never send a claim form to the policyholder, either. If AGFIL has sent the claim to the policyholder, it means Europ Assist has sent this claim to AGFIL. If AGFIL never sends the claim to the policyholder, then Europ Assist never sends a claim to AGFIL.

Theorem 4.3.2.3 *Let $\mathbb{A}$ be a set of actions, $a, b, c \in \mathbb{A}$; Σ be an alphabet, $a, \overline{a}, b, \overline{b}, c, \overline{c} \in \Sigma$; $\mathcal{W}$ be a set of contract constraints, $((\overline{a} \vee \overline{b}) \wedge \overline{c}) \vee a \cdot b \cdot c \vee b \cdot a \cdot c$ be a contract constraint scheme, and $((\overline{a} \vee \overline{b}) \wedge \overline{c}) \vee a \cdot b \cdot c \vee b \cdot a \cdot c \in \mathcal{W}$.*

Guards of this contract constraint scheme are

$$\begin{aligned}
\mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c} \vee a \cdot b \cdot c \vee b \cdot a \cdot c, a) \equiv & \Diamond \overline{b} \wedge \Diamond \overline{c} \vee (\neg b \wedge \neg c \wedge \Diamond (b \cdot c)) \vee \\
& (\Box b \wedge (\neg c \wedge \Diamond c)) \\
\mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c} \vee a \cdot b \cdot c \vee b \cdot a \cdot c, \overline{a}) \equiv & \Diamond \overline{c} \\
\mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c} \vee a \cdot b \cdot c \vee b \cdot a \cdot c, b) \equiv & \Diamond \overline{a} \wedge \Diamond \overline{c} \vee (\Box a \wedge \neg c \wedge \Diamond c) \vee \\
& (\neg a \wedge \neg b \wedge \Diamond (a \cdot c)) \\
\mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c} \vee a \cdot b \cdot c \vee b \cdot a \cdot c, \overline{b}) \equiv & \Diamond \overline{c} \\
\mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c} \vee a \cdot b \cdot c \vee b \cdot a \cdot c, c) \equiv & \Box a \wedge \Box b \\
\mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c} \vee a \cdot b \cdot c \vee b \cdot a \cdot c, \overline{c}) \equiv & \Diamond \overline{a} \vee \Diamond \overline{b}
\end{aligned}$$

Proof: we prove these six equations separately,

According to **Definition 4.3.2.1 (4.4) and (4.3)**, we have $\mathsf{G}(\overline{a}, a) \equiv 0$; $\mathsf{G}(\overline{b}, a) \equiv \Diamond \overline{b}$, $\mathsf{G}(\overline{c}, a) \equiv \Diamond \overline{c}$, $\mathsf{G}(a \cdot b \cdot c, a) \equiv \neg b \wedge \neg c \wedge \Diamond (b \cdot c)$ and $\mathsf{G}(b \cdot a \cdot c, a) \equiv \Box b \wedge \neg c \wedge \Diamond c$, respectively; according to **Definition 4.3.2.1 (4.2) and (4.1)**, we have $\mathsf{G}(\overline{a}, a) \vee \mathsf{G}(\overline{b}, a) \equiv \mathsf{G}(\overline{a} \vee \overline{b}, a)$, $\mathsf{G}(\overline{a} \vee \overline{b}, a) \wedge \mathsf{G}(\overline{c}, a) \equiv \mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c}, a)$, $\mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c}, a) \vee \mathsf{G}(a \cdot b \cdot c, a) \equiv \mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c} \vee a \cdot b \cdot c, a)$ and $\mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c} \vee a \cdot b \cdot c, a) \vee \mathsf{G}(b \cdot a \cdot c, a) \equiv \mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c} \vee a \cdot b \cdot c \vee b \cdot a \cdot c, a)$, respectively. Putting them together yields proof trees such as

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{G}((\overline{a},a)\equiv 0 \;\; \mathsf{G}((\overline{b},a)\equiv\Diamond\overline{b}}{\mathsf{G}((\overline{a},a)\vee\mathsf{G}(\overline{b},a)\equiv\Diamond\overline{b}} \quad \mathsf{G}(\overline{c},a)\equiv\Diamond\overline{c}}{\mathsf{G}((\overline{a}\vee\overline{b},a)\equiv\Diamond\overline{b}} \quad \mathsf{G}(a\cdot b\cdot c,a)\equiv\neg b\wedge\neg c\wedge\Diamond(b\cdot c)}{\mathsf{G}((\overline{a}\vee\overline{b},a)\wedge\mathsf{G}(\overline{c},a)\equiv\Diamond\overline{b}\wedge\Diamond\overline{c}} \quad \mathsf{G}(b\cdot a\cdot c,a)\equiv\Box b\wedge\neg c\wedge\Diamond c}{\mathsf{G}((\overline{a}\vee\overline{b}\wedge\overline{c},a)\equiv\Diamond\overline{b}\wedge\Diamond\overline{c}} \quad \mathsf{G}(b\cdot a\cdot c,a)\equiv\Box b\wedge\neg c\wedge\Diamond c}{\mathsf{G}((\overline{a}\vee\overline{b}\wedge\overline{c},a)\vee\mathsf{G}(a\cdot b\cdot c,a)\equiv\Diamond\overline{b}\wedge\Diamond\overline{c}\vee(\neg b\wedge\neg c\wedge\Diamond(b\cdot c))} \quad \mathsf{G}(b\cdot a\cdot c,a)\equiv\Box b\wedge\neg c\wedge\Diamond c}{\mathsf{G}((\overline{a}\vee\overline{b}\wedge\overline{c}\vee a\cdot b\cdot c,a)\equiv\Diamond\overline{b}\wedge\Diamond\overline{c}\vee(\neg b\wedge\neg c\wedge\Diamond(b\cdot c))} \quad \mathsf{G}(b\cdot a\cdot c,a)\equiv\Box b\wedge\neg c\wedge\Diamond c}{\mathsf{G}((\overline{a}\vee\overline{b}\wedge\overline{c}\vee a\cdot b\cdot c)\vee\mathsf{G}(b\cdot a\cdot c,a)\equiv\Diamond\overline{b}\wedge\Diamond\overline{c}\vee(\neg b\wedge\neg c\wedge\Diamond(b\cdot c))\vee(\Box b\wedge\neg c\wedge\Diamond c)}$$

$$\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c}\vee a\cdot b\cdot c\vee b\cdot a\cdot c,a)\equiv\Diamond\overline{b}\wedge\Diamond\overline{c}\vee(\neg b\wedge\neg c\wedge\Diamond(b\cdot c))\vee(\Box b\wedge\neg c\wedge\Diamond c)$$

According to **Definition 4.3.2.1 (4.4) and (4.3)**, we have $\mathsf{G}(\overline{a}, \overline{a}) \equiv \top$; $\mathsf{G}(\overline{b}, \overline{a}) \equiv \Diamond \overline{b}$, $\mathsf{G}(\overline{c}, \overline{a}) \equiv \Diamond \overline{c}$, $\mathsf{G}(a \cdot b \cdot c, \overline{a}) \equiv 0$ and $\mathsf{G}(b \cdot a \cdot c, \overline{a}) \equiv 0$, respectively; according to **Definition 4.3.2.1 (4.2) and (4.1)**, we have $\mathsf{G}(\overline{a}, \overline{a}) \vee \mathsf{G}(\overline{b}, \overline{a}) \equiv \mathsf{G}(\overline{a} \vee \overline{b}, \overline{a})$, $\mathsf{G}(\overline{a} \vee \overline{b}, \overline{a}) \wedge \mathsf{G}(\overline{c}, \overline{a}) \equiv \mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c}, \overline{a})$, $\mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c}, \overline{a}) \vee \mathsf{G}(a \cdot b \cdot c, \overline{a}) \equiv \mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c} \vee a \cdot b \cdot c, \overline{a})$ and $\mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c} \vee a \cdot b \cdot c, \overline{a}) \vee \mathsf{G}(b \cdot a \cdot c, \overline{a}) \equiv \mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c} \vee a \cdot b \cdot c \vee b \cdot a \cdot c, \overline{a})$, respectively. Putting

them together yields proof trees such as

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{G}(\overline{a},\overline{a}) \equiv \top \quad \mathsf{G}(\overline{b},\overline{a}) \equiv \Diamond\overline{b}}{\mathsf{G}(\overline{a},\overline{a}) \vee \mathsf{G}(\overline{b},\overline{a}) \equiv \top}}{\mathsf{G}(\overline{a} \vee \overline{b},\overline{a}) \equiv \top}\ \mathsf{G}(\overline{c},\overline{a}) \equiv \Diamond\overline{c}}{\mathsf{G}((\overline{a} \vee \overline{b},\overline{a}) \wedge \mathsf{G}(\overline{c},\overline{a}) \equiv \Diamond\overline{c}}}{\mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c},\overline{a}) \equiv \Diamond\overline{c}}\ \mathsf{G}(a \cdot b \cdot c,\overline{a}) \equiv 0}{\mathsf{G}((\overline{a} \vee \overline{b} \wedge \overline{c},\overline{a}) \vee \mathsf{G}(a \cdot b \cdot c,\overline{a}) \equiv \Diamond\overline{c}}}{\mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c} \vee a \cdot b \cdot c,\overline{a}) \equiv \Diamond\overline{c}}\ \mathsf{G}(b \cdot a \cdot c,\overline{a}) \equiv 0}{\mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c} \vee a \cdot b \cdot c \vee a \cdot b \cdot c,\overline{a}) \vee \mathsf{G}(b \cdot a \cdot c,\overline{a}) \equiv \Diamond\overline{c}}}{\mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c} \vee a \cdot b \cdot c \vee b \cdot a \cdot c,\overline{a}) \equiv \Diamond\overline{c}}$$

According to **Definition 4.3.2.1 (4.4) and (4.3)**, we have $\mathsf{G}(\overline{a},b) \equiv \Diamond\overline{a}$; $\mathsf{G}(\overline{b},b) \equiv 0$, $\mathsf{G}(\overline{c},b) \equiv \Diamond\overline{c}$, $\mathsf{G}(a \cdot b \cdot c,b) \equiv \Box a \wedge \neg c \wedge \Diamond c$ and $\mathsf{G}(b \cdot a \cdot c,b) \equiv \neg a \wedge \neg c \wedge \Diamond(a \cdot c)$, respectively; according to **Definition 4.3.2.1 (4.2) and (4.1)**, we have $\mathsf{G}(\overline{a},b) \vee \mathsf{G}(\overline{b},b) \equiv \mathsf{G}(\overline{a} \vee \overline{b},b)$, $\mathsf{G}(\overline{a} \vee \overline{b},b) \wedge \mathsf{G}(\overline{c},b) \equiv \mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c},b)$, $\mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c},b) \vee \mathsf{G}(a \cdot b \cdot c,b) \equiv \mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c} \vee a \cdot b \cdot c,b)$ and $\mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c} \vee a \cdot b \cdot c,b) \vee \mathsf{G}(b \cdot a \cdot c,b) \equiv \mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c} \vee a \cdot b \cdot c \vee b \cdot a \cdot c,b)$, respectively. Putting them together yields proof trees such as

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{G}(\overline{a},b)\equiv\Diamond\overline{a}\ \ \mathsf{G}(\overline{b},b)\equiv 0}{\mathsf{G}(\overline{a},b)\vee\mathsf{G}(\overline{b},b)\equiv\Diamond\overline{a}}}{\mathsf{G}(\overline{a}\vee\overline{b},b)\equiv\Diamond\overline{a}}\ \mathsf{G}(\overline{c},b)\equiv\Diamond\overline{c}}{\mathsf{G}(\overline{a}\vee\overline{b},b)\wedge\mathsf{G}(\overline{c},b)\equiv\Diamond\overline{a}\wedge\Diamond\overline{c}}}{\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c},b)\equiv\Diamond\overline{a}\wedge\Diamond\overline{c}}\ \mathsf{G}(a\cdot b\cdot c,b)\equiv\Box a\wedge\neg c\wedge\Diamond c}{\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c},b)\vee\mathsf{G}(a\cdot b\cdot c,b)\equiv\Diamond\overline{a}\wedge\Diamond\overline{c}\vee(\Box a\wedge\neg c\wedge\Diamond c)}}{\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c}\vee a\cdot b\cdot c,b)\equiv\Diamond\overline{a}\wedge\Diamond\overline{c}\vee(\Box a\wedge\neg c\wedge\Diamond c)}\ \mathsf{G}(b\cdot a\cdot c,b)\equiv\neg a\wedge\neg c\wedge\Diamond(a\cdot c)}{\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c}\vee a\cdot b\cdot c,b)\vee\mathsf{G}(b\cdot a\cdot c,b)\equiv\Diamond\overline{a}\wedge\Diamond\overline{c}\vee(\Box a\wedge\neg c\wedge\Diamond c)\vee(\neg a\wedge\neg b\wedge\Diamond(a\cdot c))}}{\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c}\vee a\cdot b\cdot c\vee b\cdot a\cdot c,b)\equiv\Diamond\overline{a}\wedge\Diamond\overline{c}\vee(\Box a\wedge\neg c\wedge\Diamond c)\vee(\neg a\wedge\neg b\wedge\Diamond(a\cdot c))}$$

According to **Definition 4.3.2.1 (4.4) and (4.3)**, we have $\mathsf{G}(\overline{a},\overline{b}) \equiv \Diamond\overline{a}$; $\mathsf{G}(\overline{b},\overline{b}) \equiv \top$), $\mathsf{G}(\overline{c},\overline{b}) \equiv \Diamond\overline{c}$, $\mathsf{G}(a \cdot b \cdot c,\overline{b}) \equiv 0$ and $\mathsf{G}(b \cdot a \cdot c,\overline{b}) \equiv 0$, respectively; according to **Definition 4.3.2.1 (4.2) and (4.1)**, we have $\mathsf{G}(\overline{a},\overline{b}) \vee \mathsf{G}(\overline{b},\overline{b}) \equiv \mathsf{G}(\overline{a} \vee \overline{b},\overline{b})$, $\mathsf{G}(\overline{a} \vee \overline{b},\overline{b}) \wedge \mathsf{G}(\overline{c},\overline{b}) \equiv \mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c},\overline{b})$, $\mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c},\overline{b}) \vee \mathsf{G}(a \cdot b \cdot c,\overline{b}) \equiv \mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c} \vee a \cdot b \cdot c,\overline{b})$ and $\mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c} \vee a \cdot b \cdot c,\overline{b}) \vee \mathsf{G}(b \cdot a \cdot c,\overline{b}) \equiv \mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c} \vee a \cdot b \cdot c \vee b \cdot a \cdot c,\overline{b})$, respectively. Putting them together yields

proof trees such as

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{G}(\overline{a},\overline{b})\equiv\Diamond\overline{a}\ \ \mathsf{G}(\overline{b},\overline{b})\equiv\top}{\mathsf{G}(\overline{a}\vee\overline{b},\overline{b})\equiv\top}\ \mathsf{G}(\overline{c},\overline{b})\equiv\Diamond\overline{c}}{\mathsf{G}(\overline{a}\vee\overline{b},\overline{b})\wedge\mathsf{G}(\overline{c},\overline{b})\equiv\Diamond\overline{c}}}{\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c},\overline{b})\equiv\Diamond\overline{c}}\ \mathsf{G}(a\cdot b\cdot c,\overline{b})\equiv 0}{\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c},\overline{b})\vee\mathsf{G}(a\cdot b\cdot c,\overline{b})\equiv\Diamond\overline{c}}}{\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c}\vee a\cdot b\cdot c,\overline{b})\equiv\Diamond\overline{c}}\ \mathsf{G}(b\cdot a\cdot c,\overline{b})\equiv 0}{\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c}\vee a\cdot b\cdot c,\overline{b})\vee\mathsf{G}(b\cdot a\cdot c,\overline{b})\equiv\Diamond\overline{c}}}{\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c}\vee a\cdot b\cdot c\vee b\cdot a\cdot c,\overline{b})\equiv\Diamond\overline{c}}$$

According to **Definition 4.3.2.1 (4.4) and (4.3)**, we have $\mathsf{G}(\overline{a},c)\equiv\Diamond\overline{a}$; $\mathsf{G}(\overline{b},c)\equiv\Diamond\overline{b}$, $\mathsf{G}(\overline{c},c)\equiv 0$, $\mathsf{G}(a\cdot b\cdot c,c)\equiv\Box a\wedge\Box b$ and $\mathsf{G}(b\cdot a\cdot c,c)\equiv\Box a\wedge\Box b$, respectively; according to **Definition 4.3.2.1 (4.2) and (4.1)**, we have $\mathsf{G}(\overline{a},c)\vee\mathsf{G}(\overline{b},c)\equiv\mathsf{G}(\overline{a}\vee\overline{b},c)$, $\mathsf{G}(\overline{a}\vee\overline{b},c)\wedge\mathsf{G}(\overline{c},c)\equiv\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c},c)$, $\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c},c)\vee\mathsf{G}(a\cdot b\cdot c,c)\equiv\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c}\vee a\cdot b\cdot c,c)$ and $\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c}\vee a\cdot b\cdot c,c)\vee\mathsf{G}(b\cdot a\cdot c,c)\equiv\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c}\vee a\cdot b\cdot c\vee b\cdot a\cdot c,c)$, respectively. Putting them together yields proof trees such as

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{G}((\overline{a},c)\equiv\Diamond\overline{a}\ \ \mathsf{G}((\overline{b},c)\equiv\Diamond\overline{b}}{\mathsf{G}((\overline{a},c)\vee\mathsf{G}(\overline{b},c)\equiv\Diamond\overline{a}\vee\Diamond\overline{b}}\ \mathsf{G}(\overline{c},c)\equiv 0}{\mathsf{G}((\overline{a}\vee\overline{b},c)\equiv\Diamond\overline{a}\vee\Diamond\overline{b}}}{\mathsf{G}(\overline{a}\vee\overline{b},c)\wedge\mathsf{G}(\overline{c},c)\equiv 0}\ \mathsf{G}(a\cdot b\cdot c,c)\equiv\Box a\wedge\Box b}{\mathsf{G}(\overline{a}\vee\overline{b}\wedge\overline{c},c)\equiv 0}}{\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c},c)\vee\mathsf{G}(a\cdot b\cdot c,c)\equiv\Box a\wedge\Box b}\ \mathsf{G}(b\cdot a\cdot c,c)\equiv\Box b\wedge\Box a}{\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c}\vee a\cdot b\cdot c,c)\vee\mathsf{G}(b\cdot a\cdot c,c)\equiv\Box a\wedge\Box b}}{\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c}\vee a\cdot b\cdot c\vee b\cdot a\cdot c,c)\equiv\Box a\wedge\Box b}$$

According to **Definition 4.3.2.1 (4.4) and (4.3)**, we have $\mathsf{G}(\overline{a},\overline{c})\equiv\Diamond\overline{a}$; $\mathsf{G}(\overline{b},\overline{c})\equiv\Diamond\overline{b})$, $\mathsf{G}(\overline{c},\overline{c})\equiv\top$, $\mathsf{G}(a\cdot b\cdot c,\overline{c})\equiv 0$ and $\mathsf{G}(b\cdot a\cdot c,\overline{c})\equiv 0$, respectively; according to **Definition 4.3.2.1 (4.2) and (4.1)**, we have $\mathsf{G}(\overline{a},\overline{c})\vee\mathsf{G}(\overline{b},\overline{c})\equiv\mathsf{G}(\overline{a}\vee\overline{b},\overline{c})$, $\mathsf{G}(\overline{a}\vee\overline{b},\overline{c})\wedge\mathsf{G}(\overline{c},\overline{c})\equiv\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c},\overline{c})$, $\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c},\overline{c})\vee\mathsf{G}(a\cdot b\cdot c,\overline{c})\equiv\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c}\vee a\cdot b\cdot c,\overline{c})$ and $\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c}\vee a\cdot b\cdot c,\overline{c})\vee\mathsf{G}(b\cdot a\cdot c,\overline{c})\equiv\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c}\vee a\cdot b\cdot c\vee b\cdot a\cdot c,\overline{c})$, respectively. Putting

them together yields proof trees such as

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{G}((\overline{a},\overline{c}) \equiv \Diamond\overline{a}\ \ \mathsf{G}((\overline{b},\overline{c}) \equiv \Diamond\overline{b}}{\mathsf{G}((\overline{a},\overline{c}) \vee \mathsf{G}(\overline{b},\overline{c}) \equiv \Diamond\overline{a} \vee \Diamond\overline{b}}\ \mathsf{G}(\overline{c},\overline{c}) \equiv \top}{\mathsf{G}((\overline{a} \vee \overline{b},\overline{c}) \equiv \Diamond\overline{a} \vee \Diamond\overline{b}}}{\mathsf{G}((\overline{a} \vee \overline{b}),\overline{c}) \wedge \mathsf{G}(\overline{c},\overline{c}) \equiv \Diamond\overline{a} \vee \Diamond\overline{b}}\ \mathsf{G}(a \cdot b \cdot c,\overline{c}) \equiv 0}{\mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c},\overline{c}) \equiv \Diamond\overline{a} \vee \Diamond\overline{b}}}{\mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c},\overline{c}) \vee \mathsf{G}(a \cdot b \cdot c,\overline{c}) \equiv \Diamond\overline{a} \vee \Diamond\overline{b}}\ \mathsf{G}(b \cdot a \cdot c,\overline{c}) \equiv 0}{\mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c} \vee a \cdot b \cdot c,\overline{c}) \equiv \Diamond\overline{a} \vee \Diamond\overline{b}}}{\mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c} \vee a \cdot b \cdot c,\overline{c}) \vee \mathsf{G}(b \cdot a \cdot c,\overline{c}) \equiv \Diamond\overline{a} \vee \Diamond\overline{b}}}{\mathsf{G}((\overline{a} \vee \overline{b}) \wedge \overline{c} \vee a \cdot b \cdot c \vee b \cdot a \cdot c,\overline{c}) \equiv \Diamond\overline{a} \vee \Diamond\overline{b}}$$

□

Theorem 4.3.2.4 *Let $\mathbb{A}$ be a set of actions, $a, b \in \mathbb{A}$; Σ be an alphabet, $a, \overline{a}, b, \overline{b} \in \Sigma$, $\mathcal{W}$ be a set of contract constraints, $a \cdot b \vee \overline{b}$ be a contract constraint scheme, and $a \cdot b \vee \overline{b} \in \mathcal{W}$. Guards of this contract constraint scheme are:*

$$\begin{aligned}
&\mathsf{G}(a \cdot b \vee \overline{b}, a) \equiv \neg b \wedge \Diamond b \vee \Diamond\overline{b}\\
&\mathsf{G}(a \cdot b \vee \overline{b}, \overline{a}) \equiv \Diamond\overline{b}\\
&\mathsf{G}(a \cdot b \vee \overline{b}, b) \equiv \Box a\\
&\mathsf{G}(a \cdot b \vee \overline{b}, \overline{b}) \equiv \top
\end{aligned}$$

Proof: we prove these four equations inturn.

According to **Definition 4.3.2.1 (4.3)**, we have $\mathsf{G}(a \cdot b, a) \equiv \neg b \wedge \Diamond b$ and $\mathsf{G}(\overline{b}, a) \equiv \Diamond\overline{b}$; according to **Definition 4.3.2.1(4.1)**, we have $\mathsf{G}(a \cdot b, a) \vee \mathsf{G}(\overline{b}, a) \equiv \mathsf{G}(a \cdot b \vee \overline{b}, a)$. Putting them together yields proof trees such as

$$\dfrac{\dfrac{\mathsf{G}(a \cdot b, a) \equiv \neg b \wedge \Diamond b \qquad \mathsf{G}(\overline{b}, a) \equiv \Diamond\overline{b}}{\mathsf{G}(a \cdot b, a) \vee \mathsf{G}(\overline{b}, a) \equiv \neg b \wedge \Diamond b \vee \Diamond\overline{b}}}{\mathsf{G}(a \cdot b \vee \overline{b}, a) \equiv \neg b \wedge \Diamond b \vee \Diamond\overline{b}}$$

According to **Definition 4.3.2.1 (4.4)**, we have $\mathsf{G}(a \cdot b, \overline{a}) \equiv 0$ and $\mathsf{G}(\overline{b}, \overline{a}) \equiv \Diamond\overline{b}$; according to **Definition 4.3.2.1(4.1)**, we have $\mathsf{G}(a \cdot b, \overline{a}) \vee \mathsf{G}(\overline{b}, \overline{a}) \equiv \mathsf{G}(a \cdot b \vee \overline{b}, \overline{a})$. Putting them together yields proof trees such as

$$\dfrac{\dfrac{\mathsf{G}(a \cdot b, \overline{a}) \equiv 0 \qquad \mathsf{G}(\overline{b}, \overline{a}) \equiv \Diamond\overline{b}}{\mathsf{G}(a \cdot b, \overline{a}) \vee \mathsf{G}(\overline{b}, \overline{a}) \equiv \Diamond\overline{b}}}{\mathsf{G}(a \cdot b \vee \overline{b}, \overline{a}) \equiv \Diamond\overline{b}}$$

According to **Definition 4.3.2.1 (4.3)**, we have $\mathsf{G}(a \cdot b, b) \equiv \Box a$ and $\mathsf{G}(\overline{b}, b) \equiv 0$; according to **Definition 4.3.2.1(4.1)**, we have $\mathsf{G}(a \cdot b, b) \vee \mathsf{G}(\overline{b}, b) \equiv \mathsf{G}(a \cdot b \vee \overline{b}, b)$. Putting them together yields proof trees such as

$$\dfrac{\dfrac{\mathsf{G}(a \cdot b, b) \equiv \Box a \qquad \mathsf{G}(\overline{b}, b) \equiv 0}{\mathsf{G}(a \cdot b, b) \vee \mathsf{G}(\overline{b}, b) \equiv \Box a}}{\mathsf{G}(a \cdot b \vee \overline{b}, b) \equiv \Box a}$$

According to **Definition 4.3.2.1 (4.4)**, we have $\mathsf{G}(a \cdot b, \overline{b}) \equiv 0$ and $\mathsf{G}(\overline{b}, \overline{b}) \equiv \top$; according to **Definition 4.3.2.1(4.1)**, we have $\mathsf{G}(a \cdot b, \overline{b}) \vee \mathsf{G}(\overline{b}, \overline{b}) \equiv \mathsf{G}(a \cdot b \vee \overline{b}, \overline{b})$. Putting them together yields proof trees such as

$$\dfrac{\dfrac{\mathsf{G}(a \cdot b, \overline{b}) \equiv 0 \qquad \mathsf{G}(\overline{b}, \overline{b}) \equiv \top}{\mathsf{G}(a \cdot b, \overline{b}) \vee \mathsf{G}(\overline{b}, \overline{b}) \equiv \top}}{\mathsf{G}(a \cdot b \vee \overline{b}, \overline{b}) \equiv \top}$$

□

Theorem 4.3.2.5 *Let $\mathbb{A}$ be a set of actions, $a, b, c \in \mathbb{A}$, Σ be an alphabet, $a, \overline{a}, b, \overline{b}, c, \overline{c} \in \Sigma$, $\mathcal{W}$ be a set of contract constraints, $a \cdot b \cdot c \vee b \cdot a \cdot c \vee \overline{c}$ be a contract constraint scheme, and $a \cdot b \cdot c \vee b \cdot a \cdot c \vee \overline{c} \in \mathcal{W}$. Guards of this workflow constraint scheme are*

$$\mathsf{G}(a \cdot b \cdot c \vee b \cdot a \cdot c \vee \overline{c}, a) \equiv \neg b \wedge \neg c \wedge \Diamond(b \cdot c) \vee \Box b \wedge \neg c \wedge \Diamond c \vee \Diamond \overline{c}$$
$$\mathsf{G}(a \cdot b \cdot c \vee b \cdot a \cdot c \vee \overline{c}, \overline{a}) \equiv \Diamond \overline{c}$$
$$\mathsf{G}(a \cdot b \cdot c \vee b \cdot a \cdot c \vee \overline{c}, b) \equiv \Box a \wedge \neg c \wedge \Diamond c \vee \neg a \wedge \neg c \wedge \Diamond(a \cdot c) \vee \Diamond \overline{c}$$
$$\mathsf{G}(a \cdot b \cdot c \vee b \cdot a \cdot c \vee \overline{c}, \overline{b}) \equiv \Diamond \overline{c}$$
$$\mathsf{G}(a \cdot b \cdot c \vee b \cdot a \cdot c \vee \overline{c}, c) \equiv \Box a \wedge \Box b$$
$$\mathsf{G}(a \cdot b \cdot c \vee b \cdot a \cdot c \vee \overline{c}, \overline{c}) \equiv \top$$

Proof: we prove these six equations inturn.

According to **Definition 4.3.2.1 (4.3)** and **Definition 4.3.2.1 (4.4)**, we have $\mathsf{G}(a \cdot b \cdot c, a) \equiv \neg b \wedge \neg c \wedge \Diamond(b \cdot c)$, $\mathsf{G}(b \cdot a \cdot c, a) \equiv \Box b \wedge \neg c \wedge \Diamond c$ and $\mathsf{G}(\overline{c}, a) \equiv \Diamond \overline{c}$; according to **Definition 4.3.2.1(4.1)**, we have $\mathsf{G}(a \cdot b \cdot c, a) \vee \mathsf{G}(b \cdot a \cdot c, a) \vee \mathsf{G}(\overline{c}, a) \equiv \mathsf{G}(a \cdot b \cdot c \vee b \cdot a \cdot c \vee \overline{c}, a)$. Putting them together yields proof trees such as

$$\dfrac{\dfrac{\mathsf{G}(a \cdot b \cdot c, a) \equiv \neg b \wedge \neg c \wedge \Diamond(b \cdot c) \;\; \mathsf{G}(b \cdot a \cdot c, a) \equiv \Box b \wedge \neg c \wedge \Diamond c}{\mathsf{G}(a \cdot b \cdot c \vee b \cdot a \cdot c, a) \equiv \neg b \wedge \neg c \wedge \Diamond(b \cdot c) \vee \Box b \wedge \neg c \wedge \Diamond c} \; \mathsf{G}(\overline{c}, a) \equiv \Diamond \overline{c}}{\mathsf{G}(a \cdot b \cdot c \vee b \cdot a \cdot c \vee \overline{c}, a) \equiv \neg b \wedge \neg c \wedge \Diamond(b \cdot c) \vee \Box b \wedge \neg c \wedge \Diamond c \vee \Diamond \overline{c}}$$

According to **Definition 4.3.2.1 (4.5)** and **Definition 4.3.2.1 (4.6)**, we have $\mathsf{G}(a \cdot b \cdot c, \overline{a}) \equiv 0$, $\mathsf{G}(b \cdot a \cdot c, \overline{a}) \equiv 0$ and $\mathsf{G}(\overline{c}, \overline{a}) \equiv \Diamond \overline{c}$; according to

Definition 4.3.2.1(4.1), we have $\mathsf{G}(a \cdot b \cdot c, \bar{a}) \vee \mathsf{G}(b \cdot a \cdot c, \bar{a}) \vee \mathsf{G}(\bar{c}, \bar{a}) \equiv \mathsf{G}(a \cdot b \cdot c \vee b \cdot a \cdot c \vee \bar{c}, \bar{a})$. Putting them together yields proof trees such as

$$\frac{\dfrac{\mathsf{G}(a \cdot b \cdot c, \overline{a}) \equiv 0 \quad \mathsf{G}(b \cdot a \cdot c, \overline{a}) \equiv 0}{\mathsf{G}(a \cdot b \cdot c \vee b \cdot a \cdot c, \overline{a}) \equiv 0} \quad \mathsf{G}(\overline{c}, \overline{a}) \equiv \Diamond \overline{c}}{\mathsf{G}(a \cdot b \cdot c \vee b \cdot a \cdot c \vee \overline{c}, \overline{a}) \equiv \Diamond \overline{c}}$$

According to **Definition 4.3.2.1 (4.3)** and **Definition 4.3.2.1 (4.4)**, we have $\mathsf{G}(a \cdot b \cdot c, b) \equiv \Box a \wedge \neg c \wedge \Diamond c$, $\mathsf{G}(b \cdot a \cdot c, b) \equiv \neg a \wedge \neg c \wedge \Diamond(a \cdot c)$ and $\mathsf{G}(\bar{c}, b) \equiv \Diamond \bar{c}$; according to **Definition 4.3.2.1(4.1)**, we have $\mathsf{G}(a \cdot b \cdot c, b) \vee \mathsf{G}(b \cdot a \cdot c, b) \vee \mathsf{G}(\bar{c}, b) \equiv \mathsf{G}(a \cdot b \cdot c \vee b \cdot a \cdot c \vee \bar{c}, b)$. Putting them together yields proof trees such as

$$\frac{\dfrac{\mathsf{G}(a \cdot b \cdot c, b) \equiv \Box a \wedge \neg c \wedge \Diamond c \;\; \mathsf{G}(b \cdot a \cdot c, b) \equiv \neg a \wedge \neg c \wedge \Diamond(a \cdot c)}{\mathsf{G}(a \cdot b \cdot c \vee b \cdot a \cdot c, b) \equiv \Box a \wedge \neg c \wedge \Diamond c \vee \neg a \wedge \neg c \wedge \Diamond(a \cdot c)} \quad \mathsf{G}(\overline{c}, b) \equiv \Diamond \overline{c}}{\mathsf{G}(a \cdot b \cdot c \vee b \cdot a \cdot c \vee \overline{c}, b) \equiv \Box a \wedge \neg c \wedge \Diamond c \vee \neg a \wedge \neg c \wedge \Diamond(a \cdot c) \vee \Diamond \overline{c}}$$

According to **Definition 4.3.2.1 (4.5)** and **Definition 4.3.2.1 (4.6)**, we have $\mathsf{G}(a \cdot b \cdot c, \bar{b}) \equiv 0$, $\mathsf{G}(b \cdot a \cdot c, \bar{b}) \equiv 0$ and $\mathsf{G}(\bar{c}, \bar{b}) \equiv \Diamond \bar{c}$; according to **Definition 4.3.2.1(4.1)**, we have $\mathsf{G}(a \cdot b \cdot c, \bar{b}) \vee \mathsf{G}(b \cdot a \cdot c, \bar{b}) \vee \mathsf{G}(\bar{c}, \bar{b}) \equiv \mathsf{G}(a \cdot b \cdot c \vee b \cdot a \cdot c \vee \bar{c}, \bar{b})$. Putting them together yields proof trees such as

$$\frac{\dfrac{\mathsf{G}(a \cdot b \cdot c, \overline{b}) \equiv 0 \quad \mathsf{G}(b \cdot a \cdot c, \overline{b}) \equiv 0}{\mathsf{G}(a \cdot b \cdot c \vee b \cdot a \cdot c, \overline{b}) \equiv 0} \quad \mathsf{G}(\overline{c}, \overline{b}) \equiv \Diamond \overline{c}}{\mathsf{G}(a \cdot b \cdot c \vee b \cdot a \cdot c \vee \overline{c}, \overline{b}) \equiv \Diamond \overline{c}}$$

According to **Definition 4.3.2.1 (4.3)** and **Definition 4.3.2.1 (4.4)**, we have $\mathsf{G}(a \cdot b \cdot c, c) \equiv \Box a \wedge \Box b$, $\mathsf{G}(b \cdot a \cdot c, c) \equiv \Box a \wedge \Box b$ and $\mathsf{G}(\bar{c}, c) \equiv 0$; according to **Definition 4.3.2.1(4.1)**, we have $\mathsf{G}(a \cdot b \cdot c, c) \vee \mathsf{G}(b \cdot a \cdot c, c) \vee \mathsf{G}(\bar{c}, c) \equiv \mathsf{G}(a \cdot b \cdot c \vee b \cdot a \cdot c \vee \bar{c}, c)$. Putting them together yields proof trees such as

$$\frac{\dfrac{\mathsf{G}(a \cdot b \cdot c, c) \equiv \Box a \wedge \Box b \quad \mathsf{G}(b \cdot a \cdot c, c) \equiv \Box a \wedge \Box b}{\mathsf{G}(a \cdot b \cdot c \vee b \cdot a \cdot c, c) \equiv \Box a \wedge \Box b} \quad \mathsf{G}(\overline{c}, c) \equiv 0}{\mathsf{G}(a \cdot b \cdot c \vee b \cdot a \cdot c \vee \overline{c}, c) \equiv \Box a \wedge \Box b}$$

According to **Definition 4.3.2.1 (4.5)** and **Definition 4.3.2.1 (4.6)**, we have $\mathsf{G}(a \cdot b \cdot c, \bar{c}) \equiv 0$, $\mathsf{G}(b \cdot a \cdot c, \bar{c}) \equiv 0$ and $\mathsf{G}(\bar{c}, \bar{c}) \equiv \top$; according to **Definition 4.3.2.1(4.1)**, we have $\mathsf{G}(a \cdot b \cdot c, \bar{c}) \vee \mathsf{G}(b \cdot a \cdot c, \bar{c}) \vee \mathsf{G}(\bar{c}, \bar{c}) \equiv \mathsf{G}(a \cdot b \cdot c \vee b \cdot a \cdot c \vee \bar{c}, \bar{c})$. Putting them together yields the proof trees such as

$$\frac{\dfrac{\mathsf{G}(a \cdot b \cdot c, \overline{c}) \equiv 0 \quad \mathsf{G}(b \cdot a \cdot c, \overline{c}) \equiv 0}{\mathsf{G}(a \cdot b \cdot c \vee b \cdot a \cdot c, \overline{c}) \equiv 0} \quad \mathsf{G}(\overline{c}, \overline{c}) \equiv \top}{\mathsf{G}(a \cdot b \cdot c \vee b \cdot a \cdot c \vee \overline{c}, \overline{c}) \equiv \top}$$

□

The proofs of **Theorem** 4.3.2.3, **Theorem** 4.3.2.4 and **Theorem** 4.3.2.5 are similar to the proof of **Theorem** 4.3.2.2.

In the same way, we can prove the guards of the *disable relationship* of contract constraints: $a \vee \bar{b} \vee \bar{a} \cdot b$, the guards

$$\mathsf{G}(a \vee \bar{b} \vee \bar{a} \cdot b, a) \equiv \top$$
$$\mathsf{G}(a \vee \bar{b} \vee \bar{a} \cdot b, \bar{a}) \equiv \Diamond b$$
$$\mathsf{G}(a \vee \bar{b} \vee \bar{a} \cdot b, b) \equiv \Box \bar{a}$$
$$\mathsf{G}(a \vee \bar{b} \vee \bar{a} \cdot b, \bar{b}) \equiv \top$$

The guards of the *disable relationship* constraint mean: action a can occur always; action $\bar{a}$ occurs, meaning that action b may occurs in the future; if action b occurs, then action a can not occur from now on; action $\bar{b}$ can always occur.

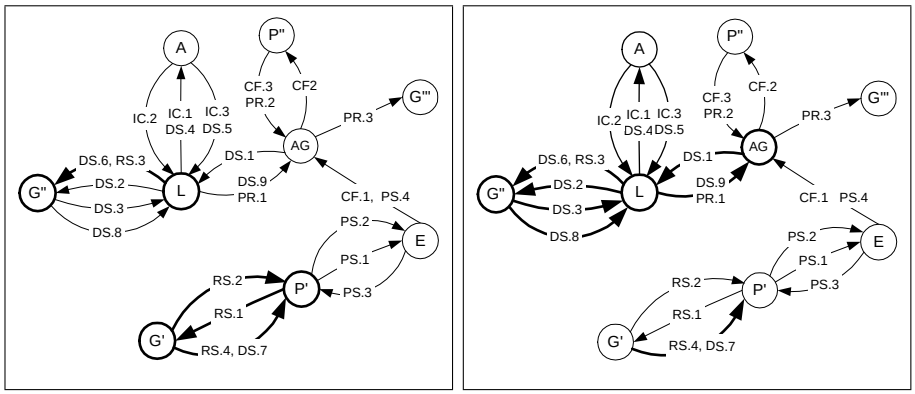
This section introduces the contract constraints and the guards of contract constraints. These provide some potential status information of actions, capturing how far that each action sequence has processes. The next section introduces the commitment graph, which deals with commitments, another important aspect of contracts.

4.4 Commitment graphs

It is widely accepted that commitments are the core of contracts. Commitments are an even more important concept, though, to specify multi-party contracts. This section presents a commitment graph that shows complex relationships among commitments. Commitment relationships are not only about a condition commitment [VS99] relationship *. For example, if a contractee first ships goods to a contractor, the contractor will pay the cost of goods later; the commitment of shipping goods is a condition to activate a commitment of payment. In our car insurance case, the relationship between *repair service commitment* and *daily service commitment* is a mixed relationship: after Lee C.S. agrees with the repair costs in *daily service commitment*, the garage can repair the car in *repair service commitment*; after the garage repairs the car and returns the invoice, *daily service commitment* will go on to execute its following actions. *Repair service commitment* and *daily service commitment* are mutually dependent on each other.

Figure 4.2 shows the commitment graph for the car insurance case. For all notes of this commitment graph, we use the following abbreviations: P' and P'' for a policyholder, AG for AGFIL, E for Euro Assist, L for Lee C.S., G', G'' and G''' for garage, and A for assessor. Each note represents a role that can be played by a contractual partner. For all edges

*We regard commitment orders as an integral of contracts. They shall formalized in **Definition** 4.4.0.7.



(a) Highlight repair service commitment C_repairService

(b) Highlight daily service commitment C_dailyService

Figure 4.2: Commitment graphs

of this graph, the following abbreviations are used: PS for C_phoneService, RS for C_repairService, DS for C_dailyService, IC for C_inspectCar, PR for C_payRepairCost, CF for C_claimForm. Each edge represents an action. Each action has one or more codes, where the first letter represents which commitments this action actually involves, the second number represents the order of a sequence actions within a commitment. Table 4.1 provides all abbreviations and codes used in this commitment graph (in Figure 4.2).

Section 4.2.2 specifies commitment C_repairService as follows:

$$(\mathsf{C\_repairService}, \mathsf{G}, \mathsf{P}, \{(\mathsf{A\_sendCar}, tr), (\mathsf{A\_estimateRepairCost}, fi), (\boxed{\mathsf{A\_agreeRepairCar}}, tr), (\boxed{\mathsf{A\_repairCar}}, fi)\})$$

The commitment involves the following actions: A_sendCar, A_estimateRepairCost, A_agreeRepairCar and A_repairCar.

We specify commitment C_dailyService as follows,

$$(\mathsf{C\_dailyService}, \mathsf{L}, \mathsf{AG}, \{(\mathsf{A\_forwardClaim}, tr), (\mathsf{A\_contactGarage}, in), (\mathsf{A\_sendRepairCost}, in), (\mathsf{A\_assignAssessor}, in), (\mathsf{A\_sendNewRepairCost}, tr), (\boxed{\mathsf{A\_agreeRepairCar}}, fi), (\boxed{\mathsf{A\_repairCar}}, tr), (\mathsf{A\_sendInvoices}, in), (\mathsf{A\_forwardInvoices}, fi)\})$$

This commitment involves following actions: A_forwardClaim, A_contactGarage, A_sendRepairCost, A_assignAssessor, A_sendNewRepairCost, A_agreeRepairCar, A_repairCar, A_sendInvoices and A_forwardInvoices.

We observe that action A_agreeRepairCar is included in commitment C_repairService as the third action and in commitment C_dailyService as the sixth action. Action

A_repairCar is included in commitment C_repairService as the fourth action, and also in commitment C_dailyService as the seventh action. Each action is indicated as an edge in a commitment graph. Information about which commitment this action is involved in and the sequence of this action for a particular commitment is recorded as edge codes. Hence, an edge for action A_agreeRepairCar has two codes: C_repairService.3 and C_dailyService.6. Our commitment graph uses edge code RS for C_repairService and edge code DS for C_dailyService; $RS.3$ and $DS.6$ both denote action A_agreeRepairCar.

Another example: the relationship between *daily service commitment* and *inspect car commitment* is an embedded relationship. If the *repair cost* is higher than a certain amount, then *daily service commitment* will trigger *inspect car commitment*. After inspecting the car, an assessor sends the new repair costs and *daily service commitment* can continue. From Section 4.2.2, we have

$$\begin{aligned}(&\mathsf{C\_dailyService}, \mathsf{L}, \mathsf{AG}, \{(\mathsf{A\_forwardClaim}, tr), (\mathsf{A\_contactGarage}, in),\\ &\qquad (\mathsf{A\_sendRepairCost}, in), (\boxed{\mathsf{A\_assignAssessor}}, in),\\ &\qquad (\boxed{\mathsf{A\_sendNewRepairCost}}, tr), (\mathsf{A\_agreeRepairCar}, fi),\\ &\qquad (\mathsf{A\_repairCar}, tr), (\mathsf{A\_sendInvoices}, in),\\ &\qquad (\mathsf{A\_forwardInvoices}, fi)\})\\ (&\mathsf{C\_inspectCar}, \mathsf{A}, \mathsf{L}, \{(\boxed{\mathsf{A\_assignAssessor}}, tr), (\mathsf{A\_inspectCar}, in),\\ &\qquad (\boxed{\mathsf{A\_sendNewRepairCost}}, fi)\})\end{aligned}$$

After all actions included in commitment C_inspectCar are finished, commitment C_dailyService can continue. We use IC for C_inspectCar; then, action A_assignAssessor has two edge codes$DS.4$ and $IC.1$, and action A_sendNewRepairCost has two edge codes $DS.5$ and $IC.3$.

A commitment graph is a directed graph consisting of a set of notes corresponding to all roles $\mathbb{R}$, a set of edges corresponding to actions and their codes, and commitment orders.

Definition 4.4.0.6 *Let $\mathbb{A}$ be a set of actions, $a \in \mathbb{A}$, $\mathbb{M}$ be a set of commitments, $m \in \mathbb{M}$, and $X = \{1, 2, \ldots\}$, a sequence function $f_{position}(a, m)$, an edge is specified as a relation from $\mathbb{A} \times \mathbb{M} \times X$*

$$edge = \bigcup_{\forall m, a \in m} \{(a, m, f_{position}(a, m)) : a \in \mathbb{A}, m \in \mathbb{M}, f_{position}(a, m) \in X\}$$

a set of all edges is

$$\mathbb{E} = \bigcup_{\forall a \in \mathbb{A}} \{edge\}.$$

Definition 4.4.0.7 *Let $\mathbb{M}$ be a set of commitments. A commitment occurrence order is specified as a relation from $\mathbb{M} \times \mathbb{M}$:*

$$order\_commitment = \{(m_1 \cdot m_2) : m_1, m_2 \in \mathbb{M}\}$$

If $m_1 \cdot m_2$ is a commitment order, we interpret it as follows: commitment m_2 is only active when commitment m_1 has been finished (i.e. only after the actions of

m_1 have occurred in the past and the commitment m_2 can be incorporated to trace back contract violations).

A set of commitment orders lists all relationships in which a commitment occurs prior to another commitment, and is specified as follows:

$$\mathbb{O} = \bigcup_{\forall x \in \mathcal{P}} \{(m_1 \cdot m_2)\}.$$

For the car insurance case, examples of the commitment orders inductively presented are the following:

$$\begin{aligned}\mathbb{O} = \{&\mathsf{C\_phoneService} \cdot \mathsf{C\_repairService}, \mathsf{C\_phoneService} \cdot \mathsf{C\_dailyService},\\ &\mathsf{C\_phoneService} \cdot \mathsf{C\_claimForm}, \mathsf{C\_dailyService} \cdot \mathsf{C\_inspectCar},\\ &\mathsf{C\_repairService} \cdot \mathsf{C\_payRepairCost}, \mathsf{C\_dailyService} \cdot \mathsf{C\_payRepairCost},\\ &\mathsf{C\_inspectCar} \cdot \mathsf{C\_payRepairCost}, \mathsf{C\_claimForm} \cdot \mathsf{C\_payRepairCost}\}\end{aligned}$$

After specification of commitment graph notes, edges, and commitment occurrence orders, the commitment graph can be specified as follows:

Definition 4.4.0.8 *Let $\mathbb{R}$ be a set of nodes, $\mathbb{E}$ be a set of edges, and $\mathbb{O}$ be a set of commitment order list. The commitment graph is defined as follows*

$$G = (\mathbb{R}, \mathbb{E}, \mathbb{O})$$

In short, a commitment graph is a visual tool to show commitment relationships in a complex multi-party contract. Besides showing a complex relationships between commitments, a commitment graph can also be used in the contract preparation stage and in the contract reactive monitoring stage.

In the contract preparation stage, it is important to know which part(s) of the contract is/are weakest. According to law [WW99], parties must have an enforceable contract to force the violative partners. In other words, the enforcement or compensation clauses should already exist in a original contract. The precise kind of remedy available to the innocent parties is predetermined to the contract. How can the parties foresee case of violation and circumstances? In the commitment graph, all actions with a "trigger" attribute and all actions with more than one code are weaker parts of a contract. For example, action A_sendCar has an "trigger" attribute. Imagine that the policyholder does not send the car to the assigned garage. However, Euro Assist, AGFIL, and Lee C.S. have worked on this claim. All repair processes have to be delayed or terminated. It is better to insert some compensation clauses in advance, so that the whole business process can be fulfilled smoothly. As another example, action A_agreeRepairCar is involved in *daily service commitment* and *repair service commitment*. If this action is not performed in time, commitments *daily service* and *repair service* both cannot continue. That would ultimately stop the whole business process. Therefore it is better to insert an enforcement or compensation clause for this action. Hence, the commitment graph is an important assistant in the contract preparation stage. It specifies what the consequences of breach in all possible cases of violation. The commitment graph provides a possibility for all parties, who usually tend to anticipate performance

they may not wish to specify in detail such penalties in advance, rather they may prefer to defer such decisions till breach arises in practice and assess its impact and hence determine the penalty in practice.

In the reactive monitoring stage, the commitment graph is also useful in helping to locate a contract violation and to find who is(are) the responsible partner(s) for a contract violation. This will be discussed in detail in Chapter 5.

Thus far we have discussed the trade process, logic relationship, and commitment graph. The next section presents our formal monitorable contract model.

4.5 Formal monitorable contract model

Now that all elements of our monitorable contract model have been presented, a formal model is provided as follows:

Definition 4.5.0.9 *Let $\mathbb{A}$ be a set of actions, $\mathbb{M}$ be a set of commitments, $\mathcal{W}$ be a set of contract constraints that are involved in a contract, $\mathcal{G}$ be a set of guards of all contract constraints, and G be a commitment graph of a contract. The monitorable contract is specified as*

$$Contract = \{\mathbb{A}, \mathbb{M}, \mathcal{W}, \mathcal{G}, G\}$$

In Section 4.2.1, Definition 4.2.1.3 defines set $\mathbb{A}$ which includes all actions in a paper contract. Section 4.2.2 defines set $\mathbb{M}$ in Definition 4.2.2.1 which covers all commitments in the multi-party contract. $\mathcal{W}$ represents all contract constraints involved in the contract which is explained in Section 4.3.1. $\mathcal{G}$ refers to the set of guards of all contract constraints which is presented Definition 4.3.2.1 in Section 4.3.2. Finally, G is a commitment graph of a ctonact which is specified in Section 4.4.

4.6 Summary

This section described our monitorable contract model, which includes the trading process, the logic relationship and the commitment graph. The trading process includes the actions and commitments. The actions indicate what each partner should play, whereas the commitments specify a guarantee between contractual partners. The logic relationship specifies the contract constraints and the guards of contract constraints. The contract constraints show the occurrence order among actions in a business process. The guard of an action for a contract constraint refers to actions that have occurred, those that have not occurred but are expected to occur, and those that should not occur in the future. The contract constraints and the guards of contract constraints provide some potential status information of actions, which captures how far each action sequence has processed. The commitment graph shows complex relationships among commitments.

Chapter 5

Monitoring Mechanism

Chapter 4 introduced our monitorable contract model. This chapter discusses how to monitor the monitorable contract in the contract fulfillment stage for realizing the pro-active monitor. In other words, given the current state of a contract execution, which actions are expected from a partner in the future, and is a contract violation likely to happen within a short time frame?

The monitoring mechanism includes a *monitoring module* and a *reactive module*. Section 5.1 explains how our monitorable contract model and our monitoring mechanism work together to detect any imminent violation in a dynamic environment. Section 5.2 explains the monitoring module of the monitoring mechanism which includes two algorithms: a *maintaining guard algorithm* and a *pro-active detection algorithm*, and a Petri Net module. The maintaining guard algorithm shows how to use guards of contract constraints to dynamically monitor business processes. The pro-active detection algorithm shows how to use guards and the deadline of actions together to trigger the reactive module. The Petri Net module traces the contract execution. We discuss the reactive module of the monitoring mechanism in Section 5.3. From a pro-active monitoring perspective, we explain a reminding module and a warning module of the reactive module in Section 5.3.1. From a reactive monitoring perspective, we illustrate how to detect responsible partners after a contract violation in Section 5.3.2. We summarize our discussion in Section 5.4.

5.1 How the monitoring mechanism and the monitorable contract model work together

The monitoring mechanism is a dynamic mechanism for contract monitoring, and consists of the *monitoring module* and the *reactive module*. The *monitoring module* includes the maintaining guard algorithm, the pro-active detection algorithm, and the Petri Net model. The maintaining guard algorithm and the proactive detection algorithm are used to realize pro-active monitoring, whereas the Petri Net model provides the overview of the business process execution, which mainly contributes to the reactive monitoring. The *reactive module* has four submodules: reminding, warning, tracing and compensating. The reminding and warning modules finish the pro-active monitoring; the trace and compensation executes the reactive mon-

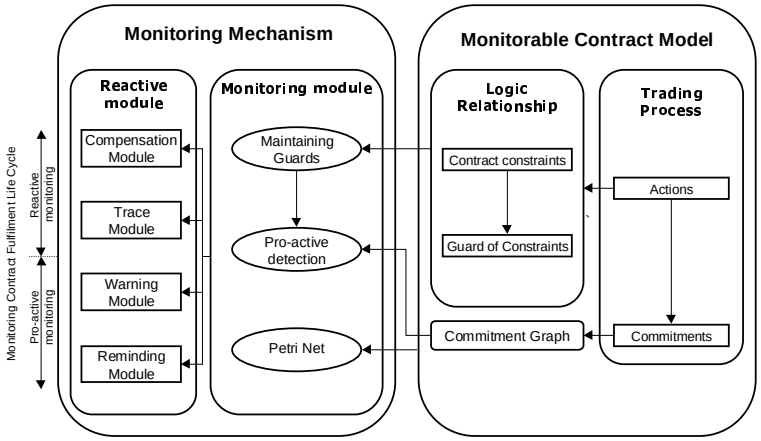


Figure 5.1: Monitoring mechanism and monitorable contract model

itoring.

Figure 5.1 shows the monitorable contract model and the monitoring mechanism, which work together to realize the pro-active monitor at the contract fulfillment stage. According to the monitorable contract model, the monitoring module observes occurred, that have activities and captures any relevant information that arises during the fulfillment of the contract by using the monitoring module. The reactive module receives the triggered information based on the output of the monitoring module and the monitorable contract, and sends a relevant message (such as warning and reminding) as a response. The details of the monitoring mechanism will be explained in turn in the following sections.

5.2 Monitoring module

The monitoring module of the monitoring mechanism includes the maintaining guard algorithm, the pro-active detection algorithm and the Petri Net model. The maintaining guard algorithm and the pro-active detection algorithm dynamically trace which actions have occurred, which have not occurred but are expected to occur, and which may never occur. The maintaining guard algorithm and the pro-active algorithm also provide a possibility for reminding and warning messages to be sent before anomalous actions occur. The Petri Net provides an overview of the business process. In the following sections, the car insurance case (details see Appendix A) is used to explain our algorithms and the Petri Net model. Sections 5.2.1, 5.2.2 and 5.2.3 present the maintaining guard algorithm, the pro-active detection algorithm and the Petri Net model, repectively.

5.2.1 Algorithm for maintaining guards

Where guards are essential to the use of temporal logic in many computer science and information systems applications, when used in a executable context, it is necessary to define algorithms that allow for efficient the machine execution and maintenance of the the guards and corresponding logic state. While some

straightforward approaches exist, as desribed partially in [Sin97] this is insufficient for execution.**Algorithm 1** provides an explicit representation of such a guard maintainance algorithm. Even in the simple approach that is described there, it is apparent that there are opportunities for errors to be made. In addition, the algorithm is insufficient, and **Algorithm 2** therefore describes a more complex approach that enables pro-active multi-party contract monitoring. It is imporant to note that in this multi-party contract case, all parties are independent and cannot be forced to perform actions, such as is the case in agent systems. In particular the system cannot assume that contract violations are mere accidents and cannot assume full benevolence in all cases.†

When an action a is attempted, its guard is evaluated, where evaluation usually means checking if the guard evaluates to $\top$. Based on sections 4.3.1 and 4.3.2(*contract constraints* and *guards of contract constraints*, respectively), we know that actions can occur, can never occur, or have not yet occurred. If the guard of action a is satisfied, action a is executed; if it is 0, then action a is rejected; otherwise, the partner of perform action a is made to wait. Whenever action a occurs, notification is sent to each pertinent partner who performs action b, whose guards are updated accordingly. If the guard of action b becomes $\top$, then action b is allowed; if it becomes 0, action b is rejected; otherwise, the partner of perform action b is made to wait longer, and so on. In this algorithm, $\mathbb{A}$ is a set of all actions; $\Sigma = \{a, \overline{a} : a \in \mathbb{A}\}$ is set of actions and their complement, U and V are the temp set of actions and their complement, $guard[u]$ and $guard[v]$ are the temp storage of guards. We dynamically update our guards using **Algorithm 1**.

For example, the guards of the contract constraint scheme 1 (see Theorem 4.3.2.2) are

$$\mathsf{G}(\overline{a} \wedge \overline{b} \vee a \cdot b, a) \equiv \neg b \wedge \Diamond b$$
$$\mathsf{G}(\overline{a} \wedge \overline{b} \vee a \cdot b, \overline{a}) \equiv \Diamond \overline{b}$$
$$\mathsf{G}(\overline{a} \wedge \overline{b} \vee a \cdot b, b) \equiv \Box a$$
$$\mathsf{G}(\overline{a} \wedge \overline{b} \vee a \cdot b, \overline{b}) \equiv \Diamond \overline{a}$$

if action a is attempted and action b has not yet happened, action a's guard evaluates to $\top$; consequently, action a is allowed and a notification $\Box a$ is sent. Upon receipt of this notification, action b's guard is simplified from $\Box a$ to $\top$. Now if action b is attempted, it can happen immediately. If action b is attempted first, it must wait because its guard is $\Box a$. Sometime later if $\Box a$ received, its guard is simplified to $\top$.

Example 5.2.1.1 *A relationship between actions* A_sendCar *and* A_estimateRepairCost *belongs to constraint scheme 1: the policyholder must send the car to the garage which initates the garage estimates the repair cost.*

- *If action* A_sendCar *is attempted and action* A_estimateRepairCost *has not yet happened, the guard of action* A_sendCar *evaluates to* $\top$*, consequently, action* A_sendCar *is allowed and a notification of the occurrence of action*

†This paragraph, and similarly marked paragraphs in chapters 3 and 4 have been added or expanded to add further clarification following a LOWI procedure in which LOWI found the author acted with culpable negligence in omitting a proper explanation as to the origins of the logic used.

Algorithm 1 Maintaining guards

```
U = ∅;
while Σ − U ≠ ∅ do
  take next u out of Σ − U;
  guard[u] = G(constraint, u)
  if guard[u] ≡ ⊤ then
    do(u);
    U = U ∪ {u, ū};
    V = Σ − U;
    while V ≠ ∅ do
      take any v out of V;
      guard[v] = G(guard[u], v);
      guard[v̄] = G(guard[u], v̄);
      V = V − {v, v̄};
    end while
  else if guard[u] ≡ 0 then
    reject occurrence of u;
  else
    send reminding or warning messages;
  end if
end while
```

A_sendCar *is sent. Upon receipt of this notification, the guard of action* A_estimateRepairCost *is simplified from* □A_sendCar *to* ⊤. *If action* A_estimateRepairCost *is attempted now, it can happen immediately.*

- *If action* A_estimateRepairCost *is attempted first, it must wait because its guard is* □A_sendCar. *Sometimes later if* □ A_sendCar *received, its guard is simplified to* ⊤.

The maintaining guard algorithm shows how to use guards to dynamically monitor business processes. It makes it possible to send the pertinent reminding message or warning message during the fulfillment of the contract.

5.2.2 Algorithm for pro-active detection

Our approach uses guards of contract constraints for proactive monitoring purposes. We will explain how to use guards and the deadline of actions together to trigger the reactive module. The pro-active detection algorithm is specified in **Algorithm 2**. w denotes a contract constraint scheme. a and b are actions, $a, b \in w$. $a.t$ indicates the deadline of action a. We use Petri Net to overview a business process. The *places* correspond to states or each party, and *transitions* correspond to actions of different parties, M_0 being an initial marking of Petri Net.

When an action is attempted, if the guard of the action evaluates to ⊤, this action can occur immediately, the rest of actions will be notified and their guards will be updated. Let $f_w(a)$ be a function that gives warning time of *action* a. If $f_w(a)$ has passed and the action has not yet occurred, the warning module will be

triggered and a warning message will be sent to the parties involved. The time boundary between triggering the reminding module and the warning module can be changed in different applications.

Example 5.2.2.1 *When the guard of* A_sendCar *and* A_estimateRepairCost *for* A_sendCar *trying is evaluated as* $\top$*, the policyholder does not send his car to the garage before* f_w(A_sendCar)*. The policyholder will receive a warning message. The garage will be reminded by "do not do anything yet", as well as AGFIL will be reminded when the guard of* A_forwardClaim *and* A_contactGarage *for* A_forwardClaim *trying and the guard of* A_sendClaimForm *and* A_returnClaimForm *for* A_sendClaimForm *trying are* $\top$*, or Lee C.S. will be reminded when the guard of* A_contactGarage *and* A_sendRepairCost *for* A_contactGarage *trying is evaluated as* $\top$ *which depends on the time.*

Algorithm 2 Pro-active detection

```
if G(w, a) ≡ ⊤ then
  while ¬done(a) ∧ f_w(a) < t ≤ a.t do
    warn(a.sender)    //the potential violation.
    ∀b ∈ {w − {a}} remind(b.sender)    //the potential 'victim' of
                                       //the violation.
    for all G(w', a') ≡ ⊤ do
      ∀c ∈ {w'} remind(c.sender)    //remind all executing partiners
    end for
  end while
  if done(a) ≡ TRUE then
    update(M_0)
    Call Algorithm 1
    remind(b)
  end if
end if
```

5.2.3 Petri Net

As we mention before, Petri Net is used to overview a business process[Rei92] [DE95] [Xu02a] [Xu02b], the *places* correspond to states of each party, and *transitions* correspond to actions of different parties, M_0 being an initial marking of Petri Net.

Let $\mathcal{S}$ be a set of possible states of the contract, $\mathbb{A}$ be a set of actions in the contract, and $\mathcal{F}$ be a set of arcs. A Petri Net N is specified as a triple: $N = (\mathcal{S}, \mathbb{A}, \mathcal{F})$ [DE95]; it consists of states from $\mathcal{S}$, actions from $\mathbb{A}$ and arcs from $\mathcal{F}$.

A net N is T-*restricted* iff for each action $a \in \mathbb{A}$ there exist states $s_1, s_2 \in \mathcal{S}$, such that $(s_1, a) \in \mathcal{F}$ and $(a, s_2) \in \mathcal{F}$ hold. A net is *finitely-branching*, iff for each action $a \in \mathbb{A}$ the set $s_1 \in \mathcal{S} : (s_1, a) \in \mathcal{F} \vee (a, s_1) \in \mathcal{F}$ is finite.

A function $M :\rightarrow \mathbb{N}$ is a marking of N. N is a T-restricted and finitely-branching net, and M_0 is a finite marking of N.

Example 5.2.3.1 *Figure 5.2 presents the procedure of the car insurance case.*

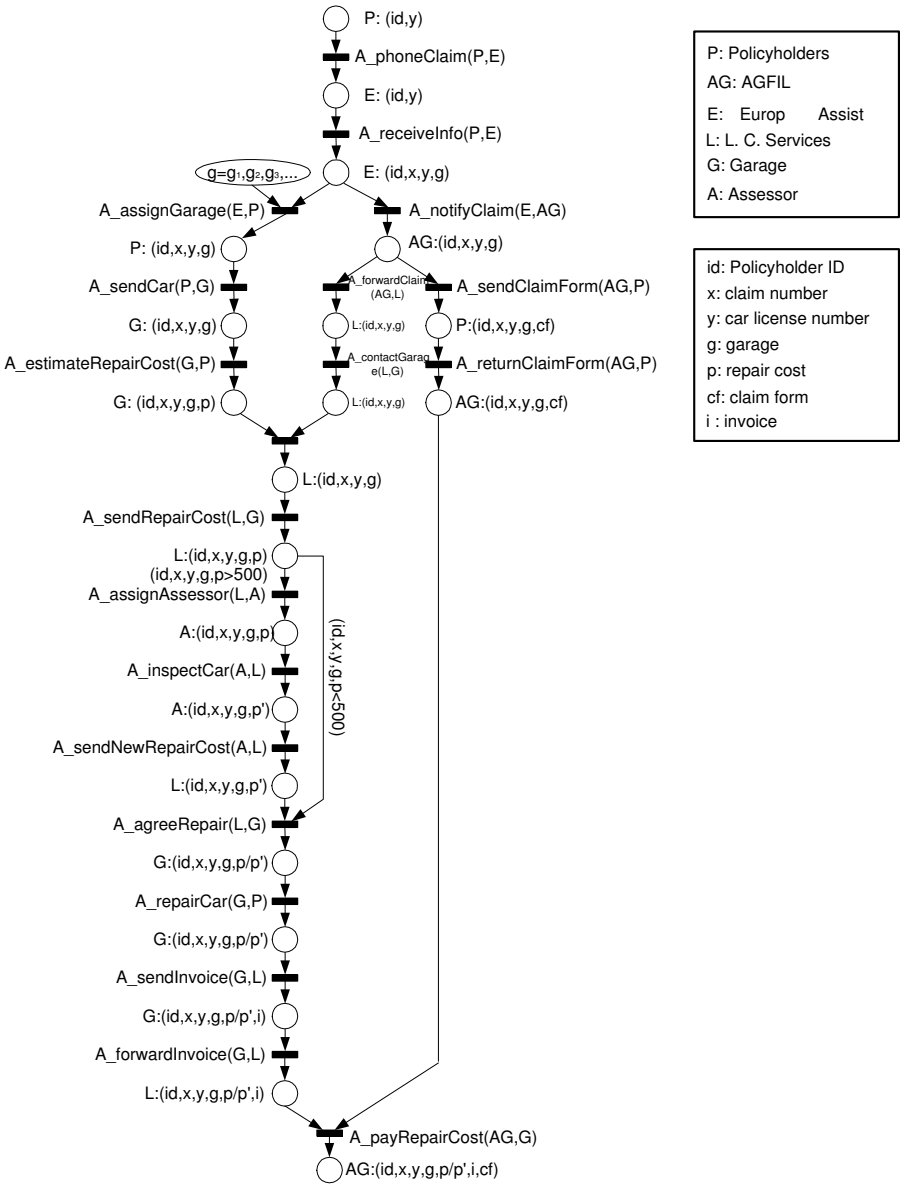


Figure 5.2: The Petri net of the car insurance case

The above sections explain our monitoring approach, which include a monitorable contract model and the monitorable module of the dynamic monitoring mechanism. Avoidance and anticipation of violations are important features for our monitoring system. The next sections will discuss reactive monitoring module.

5.3 Reactive module

The *reactive module* has four submodules: reminding, warning, tracing, and compensating modules. They respectively support the anticipation and avoidance of anomalous action occurrence at the *pro-active monitoring stage* in Section 5.3.1, and the detection and compensation of anomalous actions occurrence at the *reactive monitoring stage* in Section 5.3.2.

5.3.1 Reminding and warning module

Before an anomalous action occurrence, we should uncover situations where an anomalous action is likely to occur. We call this process an *anticipation* process. At the same time, we should reduce or eliminate the likelihood of an anomalous action. We call this process an *avoidance* process. The anticipation process and the avoidance process are both the part of the pro-active monitoring process.

Pro-active monitoring is domain-independent monitoring. To realize anticipation of an anomalous action occurrence in our car insurance case. We send reminding messages to the pertinent partners who then perform the appropriate following actions after an action occurrence. To avoid an anomalous action occurrence before the deadline of the prescribed action, we send a warning message that could include the consequence of disobedience or invoke the enforcement when it exists. Most of the time, this kind of reminding and warning function, which reminds some careless partners and points out the consequences of disobedience to the relevant partners, is useful for anticipation and avoidance of some anomalous actions. In some complex domains or applications, specific mechanisms or technologies could be explored. For example, in an agent-based e-marketplace, it is necessary to ascertain whether an agent sends a same request after accepting a bid of the request again. This behavior will cause some potential disputations (e.g. with regard to a payment issue and resource waste). As another example, in an exception-handling system of an open e-marketplace of contract net software agents [CKRa00], avoidance of agent death exceptions is realized by keeping track of agent reliability statistics (as a function of mean time between failures) and helping agents to use them when making task assignment decisions.

Although the specific reminding and warning mechanisms and technologies are not the main concern of our research, we provide a reminding and warning module, which is a message sending approach, to realize our pro-active monitoring using our monitorable contract model. We emphasize that the anticipation process and the avoidance process are domain-independent processes. How to specifically realizing pro-active monitoring depends on the application. Our research provides trigger information for pro-active monitoring of a business process. The next section explores how to detect responsible partners for a contract violation and how to compensate partners after a contract violation.

5.3.2 Detection and compensation violation scenarios

In a multi-party contractual business process, an anomalous action is sometimes not detected until other parties have already performed many actions. Retrieval of certain activities of different parties is thus necessary. The costs of non-conforming actions or anomalous actions need to be compensated. Sometimes the compensation

function is optional. At a minimum, the other parties need to be informed of the detection of a contract violation in other to prevent further costs.

In a multi-party business process, the contract violation can be caused by a series of actions that should have occurred before and involve direct and indirect contractual parties. This raises the problem of finding all responsible partners for the contract violation. The most common detection process is to retrieve all actions that have already occurred. Although it is a solution, this process is rather inefficient. In paper [GVA01], a Petri Net is used to describe the business process. A compensation graph is basically a contrary Petri Net, which starts from a violation point and stops at a safe point of the workflow. This approach is available for a workflow system with a certain safe point. Our approach is more general and does not depend on Petri Net or workflow safe points. Although we also use a Petri Net to model a business process, and a contrary Petri Net to demo a detection tree, it is not required in a particular implementation. The main point of our approach is that use of commitment graphs optimizes the detection and compensation process. Figure 5.3 shows the process of detecting responsible partners of a contract violation.

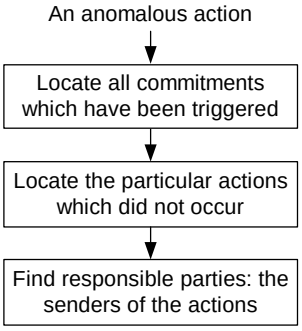


Figure 5.3: The process of detecting responsible partners

The following sections use three scenarios of the car insurance case to explain the issues arising in the performance of a multi-party contract and to demonstrate how to detect responsible partners using the commitment graph.

Example 5.3.2.1 First scenario
In the car insurance case, when Lee C.S. contacted the garage, the garage did not send the repair cost to Lee C.S. We need to know who is responsible for this mistake. Action A_contactGarage *belongs to the daily service commitment. In the commitment graph, we specify the order of commitments, and we know that there exists a mixed relationship between the daily service commitment and the repair service commitment. Thus, the phone service commitment and the repair service commitment need to be checked. In the phone service commitment, the assigned garage should be the same garage that was contracted by Lee C.S. Actions* A_notifyClaim *(*$PS4$*) and* A_assignGarage *(*$PS3$*) need to be retrieved. In the repair service commitment, the detect process needs to check whether action* A_estimateRepairCost *(*$RS2$*) and action* A_sendCar *(*$RS1$*) have been completed. If not, the responsible partner refers to the garage or the policyholder, respectively. The process of detecting this scenario is shown in Figure 5.4.*

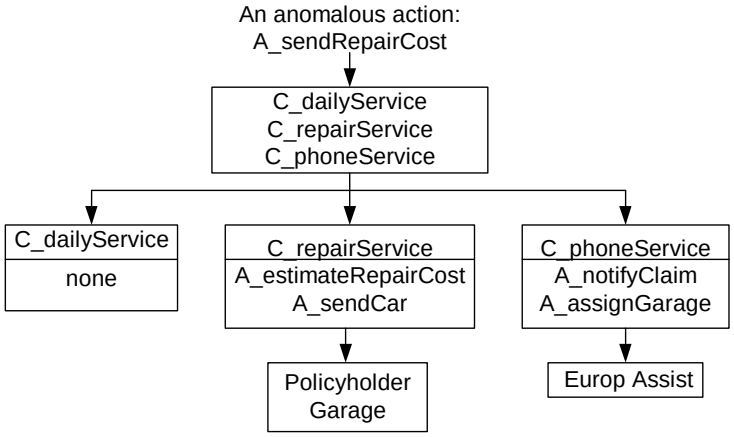
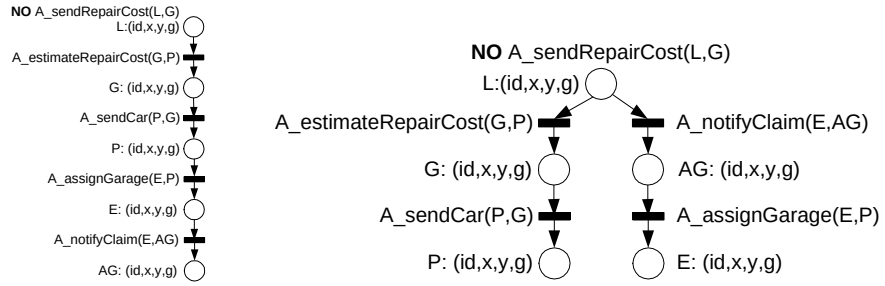


Figure 5.4: The detecting process for the first scenario

Without using the commitment graph, based only on Petri Net, we can simply obtain a compensation graph in Figure 5.5(a). The compensation graph is a deep tree; using the commitment graph can reduce levels of the detect tree.



(a) The detect tree for the first scenario

(b) The optimized detect tree for the first scenario

Figure 5.5: The detect tree for the first scenario

Example 5.3.2.2 Second scenario
The second scenario deals with another contract violation. The policyholder sent the car to the assigned garage. After the prescribed days, the policyholder finds that the garage did not repair his/her car at all.

Action A_repairCar *belongs to the repair service commitment and the daily service commitment. According to the commitment graph, the phone service commitment, the daily service commitment and the inspect service commitment could be involved before the contract violation. The process of detecting this scenario is shown in Figure 5.6.*

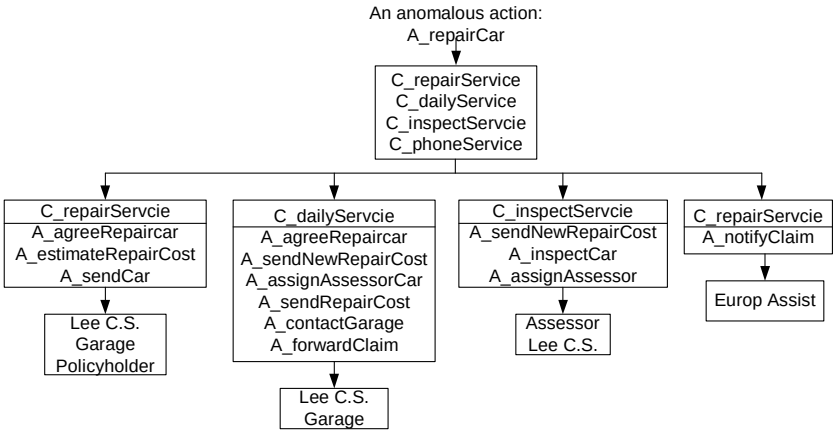


Figure 5.6: The detecting process for the second scenario

In the repair service commitment, action A_repairCar *has* $RS4$ *code. In this repair service commitment,* $RS3$ *and* $RS2$ *need to be retrieved (*$RS3$ *denotes action* A_agreeRepairCar *and* $RS2$ *denotes action* A_estimateRepairCost*).* $RS1$ *indicates action* A_sendCar*, which we knew the policyholder had done, so action* A_sendCar *do not need to retrieve.*

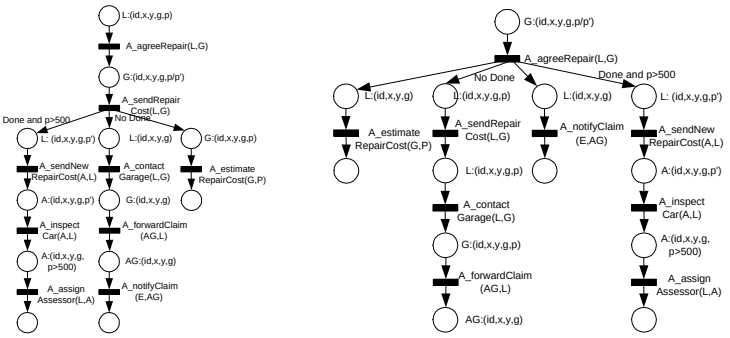
In the daily service commitment, action A_repairCar *has* $DS7$ *code. In this commitment,* $DS6$ *(action* A_agreeRepairCar*),* $DS5$ *(action* A_sendNewRepairCost*),* $DS4$ *(action* A_assignAssessor*),* $DS3$ *(action* A_sendRepairCost*),* $DS2$ *(action* A_contactGarage*) and* $DS1$ *(action* A_forwardClaim*) have to be retrieved. Because both* $DS6$ *and* $RS3$ *denote action* A_agreeRepairCar*, this action is the first action that needs to be retrieved in both the repair service commitment and the daily service commitment. Hence, this action will be retrieved first.*

In the inspect service commitment, $IC3$ *(action* A_sendNewRepairCost*),* $IC2$ *(action* A_inspectCar*) and* $IC1$ *(action* A_assignAssessor*) will be retrieved if the estimated repair cost exceeds a certain amount. From the commitment graph, we know both* $IC3$ *and* $DS5$ *denote action* A_sendNewRepairCost*, and both* $IC1$ *and* $DS4$ *indicate action* A_assignAssessor*. If* $DS5$ *(action* A_sendNewRepairCost*) and* $DS4$ *(action* A_assignAssessor*) exist, then the inspect service commitment needs to be retrieved; otherwise, the inspect commitment does not exist.*

After we have retrieved all actions in the daily service commitment, we are still unable to find who is responsible for the violation. The phone service commitment will be checked. In the phone service commitment, $PS4$ *(action* A_notifyClaim*) needs to be retrieved.*

To use the commitment graph to detect the responsible partners, we must first of all check whether the relevant commitments have been completed. For all uncompleted commitments, we check all actions of the commitment.

Example 5.3.2.3 Third scenario
The section describes another scenario of the car insurance case. After the garage sent an invoice to Lee C.S., the garage did not receive the repair cost from AGFIL. AGFIL is supposed to pay the repair cost once it receives the invoice from Lee C.S.



(a) The detect tree for the second scenario

(b) The optimized detect tree for the second scenario

Figure 5.7: The detect tree for the second scenario

and the completed claim form from the policyholder. There are two commitments involved: the daily service commitment (DS is an abbreviation of the daily service commitment) and the claim form commitment (CF is an abbreviation of the claim form commitment). The process of detecting this scenario is shown in Figure 5.8.

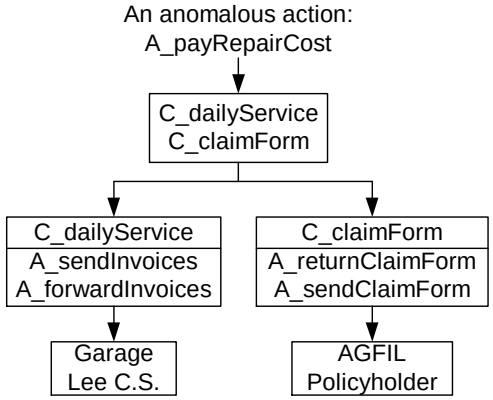


Figure 5.8: The detecting process for the third scenario

In the daily service commitment, because action A_sendInvoices *(DS*8 *as an abbreviation) has already occurred, only action* A_forwardInvoices *(DS*9*) needs to be checked; in other words, we need to check whether Lee C.S. forwarded the invoice to AGFIL.*

In the claim form commitment, actions A_returnClaimForm *(CF*3 *as an abbreviation) and* A_sendClaimForm *(CF*2 *as an abbreviation) need to be retrieved. There are three possibilities:*

i) The policyholder is the responsible partner who did not return the claim form,

ii) AGFIL is the responsible partner who did not send the claim form to the

policyholder,

iii) AGFIL received the claim form from the policyholder but did not pay the repair cost to the garage in time.

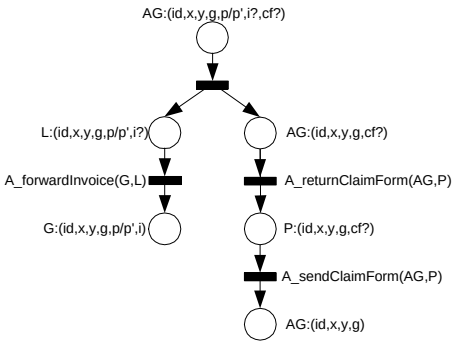


Figure 5.9: The detect tree of the car insurance case

In this scenario, the detect tree (in Figure 5.9) did not change after use the commitment graph. Hence using commitment graph, at the worst case, did not help.

Examples 5.3.2.1, 5.3.2.2 and 5.3.2.3 present how to detect the responsible partners for the contract violations. Use of the commitment graph allows optimization the detection process.

5.4 Summary

This chapter demonstrated how our monitoring mechanism and our monitorable contract model work together to dynamically monitor the execution of a business process. For the purposes of pro-active monitoring, we provide the maintaining guard algorithm and the pro-active detection algorithm, which capture some signals that can trigger the remind module and warning module of the reactive module to anticipate and avoid anomalies. For the purposes of reactive monitoring, we demonstrated an approach that makes it possible to detect the responsible partner(s) for a contract violation. This approach uses the commitment graph to improve the efficiency of the detection process.

Chapter 6

A Framework for Monitorable Contract Fulfillment

Chapters 4 and 5 proposed our monitorable contract model and monitoring mechanism. E-contracting is widely applied to e-marketplaces. Currently, there are more and more multi-party business processes involved in e-marketplaces. We give two examples to show not only the simple buy-sell kind of the business process can be done in the e-marketplace but also multi-party cases that involve more than only buy-sell relationships.

In paper [Del00], Dellarocase mentioned a financial service in an open marketplace. An electronic investor agent A is interested in locating and forming a virtual partnership with another investor agent B and a reliable stock intelligence agent C. The idea is that A and B will exclusively hire the services of agent C for a minimum time interval T. A and B will jointly pay the "salary" of agent C. A is prepared to cover up to 2/3 of the salary in exchange for getting priority in the handling of its requests.

In last minute deals, the discount sometimes is one package. It means that it is a very cheap deal for a customer, however, it may offer something more than the customer wanted. For instance, there are one package with two tickets for flying China. However, the customer may only want one ticket, it is cheap to find another customer to buy the package together. In e-marketplace, this process is finished easily involving more than two parties.

This chapter investigates how to adapt our monitorable contract model and monitoring mechanism to an e-marketplace. First, we investigate monitoring requirements for different infrastructure e-markets. We then discuss the *monitorability* of different e-market infrastructures. Monitorability refers to whether sufficient information and monitoring points are provided to a monitoring system. There are third concerns in this chapter:

- Our monitorable contract should also work at different e-market infrastructures.
- Our monitorable contract works for multi-party application in the e-market.

- Monitoring function is very important, but it is an optional choice in some cases. Thus, our contract module should work in different monitoring requirements.

Based on monitoring requirements and an analysis of different e-market infrastructures, a two-level monitoring framework is proposed.

Section 6.1 presents our two-level monitoring framework and explains the purpose of each level. The reference architecture of an agent-based e-marketplace is elaborated in Section 6.2. We end the chapter with a summary.

6.1 A two-level monitoring framework

This section presents a two-level framework. Section 6.1.1 explains why two levels of monitoring process specification are required. Then the framework is presented, and we subsequently illustrate how it can support all monitoring requirements (which are depicted in Chapter 1). Sections 6.1.2 and 6.1.3 present the structure of a central monitoring level and the structure of a local monitoring level, respectively.

6.1.1 The necessity of two-level monitoring

Different infrastructures exist for e-markets. E-market infrastructures can roughly be classified as central control e-markets [CKRa00] [DWX01] and self-regulated e-markets [DDM01]. At a centralized e-market, a central monitoring system can be employed, which is not possible at a decentralized e-market. A local level of monitoring is feasible for both e-market systems – hence, the need for a framework with a central monitoring level and a local monitoring level (as shown in Figure 6.1).

The central monitoring level features overall monitoring on behalf of contractual parties, whereas the local monitoring level operates on behalf of a single party, adjusting its monitoring request for different business processes. All parties use their local monitoring level to automatically update their monitoring information.

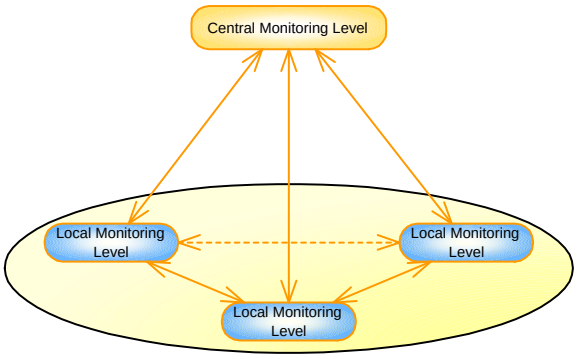


Figure 6.1: Two-level framework

With regard to agents, different kinds may be distinguished. Some agents are more independent than others; some use a remedial mechanism that might return

business processes to a normal course of action after the occurrence of anomalous behavior. For example, when an agent changes business priorities of different business processes, it first deviates from prescribed behavior for execution of another higher priority business process. Later it compensates this action and returns to normal behavior for the first business process. The agent might not want this kind of behavior to be monitored by a central monitoring agent and to receive possible punishment.

In addition, agents in and outside a society e-market can together finish a business process. It is impossible for a society central monitoring agent to monitor those outside agents, but it is necessary for the business process to be monitored by the local monitoring level. The question of how to keep a balance between central monitoring and local monitoring, and how to realize a intention of effective monitoring is complex, and depends on user requirements and implementation possibilities.

All things considered, the central monitoring level is important for multi-party business processes to collect overall monitoring information, and to arbitrate in and resolve conflicts. The local monitoring level is necessary for flexible adaptation to different e-market infrastructures. The local monitoring level makes it possible to keep monitorability in case the central monitoring level is invalid or does not exist.

6.1.2 The central monitoring level

Based on the above analysis of monitoring requirements, there are a log file, three knowledge bases and four basic function modules in the central monitoring level, as shown in Figure 6.2. They are log file, contract repository, reputation repository, disputation repository, trace module, detect module, enforcement module, and compensation module.

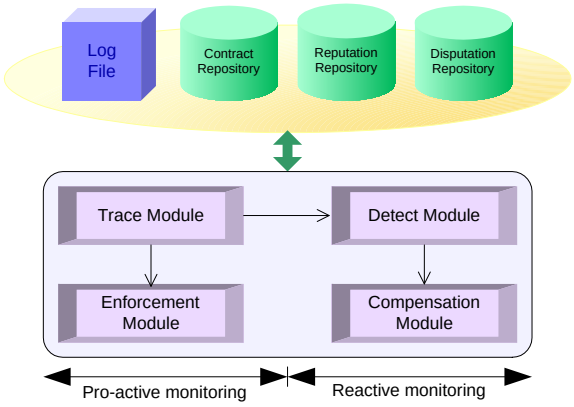


Figure 6.2: Structure of central monitoring

The functions of the four knowledge bases and a log file are as follows:

- A *contract repository* stores all monitorable e-contracts that need to be executed in the e-market.
- Based on a contract from the contract repository and a log file in which all operations are recorded, a *contract performance state* can be deduced.

- A *reputation repository* records the historical status of contract fulfillment, or the use of common financial services (e.g. a bank or a credit card company) as a reputation reference.
- A *disputation repository* stores all unresolved conflicts for human-involved resolution.

The central monitoring level uses a global view to monitor an e-market. In particular, in multi-party contract business processes, the central monitoring party can help the owner of the overall business process to monitor other parties that are involved but do not have direct contact with the main owner. The four basic function modules include a *trace module*, a *detection module*, an *enforcement module* and a *compensation module*. Our monitoring mechanism which is proposed in Chapter 5 can be configured into these four module.

Based on the contract, the contract performance state and other monitoring information, the *trace module* can indicate where the business process is, what needs to be done next, and can signal the enforcement module after it finds non-conforming actions. In a particular implementation, a Petri Net model of a business process could be an option. The Petri Net model refers to Section 5.2.3.

The *detect module* ensures that any anomalous action or non-compliant action can be exactly located. It can single out the detected result to the compensation module and send unsolved conflicts to a disputation repository. This detect module includes the maintaining guard algorithm and the pro-active detection algorithm, which are parts of our monitoring mechanism. The detect module refers to Sections 5.2.1, 5.2.2 and 5.3.2.

The *enforcement module* enables the system to respond to the trace result, to realize avoidance or anticipation of anomalous actions. Reminding or warning messages can be sent. For non-conforming actions, an enforcement mechanism can be trigged to execute the prescribed action. The implementation of the enforcement module is domain-dependent and in accordance with the e-contract. The enforcement module refers to Section 5.3.1.

The *compensation module* has the capability of compensating or undoing an action that should occur, but did not occur before or the enforcement failure. The implementation of the compensation module is domain-dependent and in accordance with the e-contract as well.

6.1.3 The local monitoring level

The local monitoring level has the same functions and structure as the central monitoring level. Because the local monitoring level gets all information only from other parties that directly contact the agent, there are some differences in certain modules. The contract repository stores only those contracts involving the local agent. The reputation repository can use common financial services or create local historical reputation information. The function of the enforcement module is limited to sending reminding and warning messages to other parties for avoidance and anticipation intentions. In addition, the forward monitoring information module can only be used when the local monitoring level works together with a central monitoring level.

This section proposed our two-level framework, and for each level, gave the detailed structure of the monitoring module for each level. The next section will introduce a reference architecture for our two-level framework.

6.2 Reference architecture

We present a reference architecture in Figure 6.3 to demonstrate how our two-level monitoring framework can be applied to an agent-based e-market. We use a trusted third party – the *Central Monitoring Agent* – in our monitoring system. The monitorable contract will be used by all agents, either normal or central monitoring agents. All agents have the *monitoring module* and the *reactive module* as well [Xu02a]. The monitoring module includes a *trace module* and a *detection module*; the reactive module includes an *enforcement module* and a *compensation module*.

In addition to the central monitoring and reactive modules, the central monitoring agent also has three repositories as its knowledge base and a log file. These three repositories are: a *contract repository* to store all contracts; a *reputation repository* to record all historical status of contract fulfillment for each partner; and a *disputation repository* to store unsolvable disputations as evidence in the human-involved deal. Particularly, the central monitoring agent provides the common time standard to give each occurred action a time-stamp.

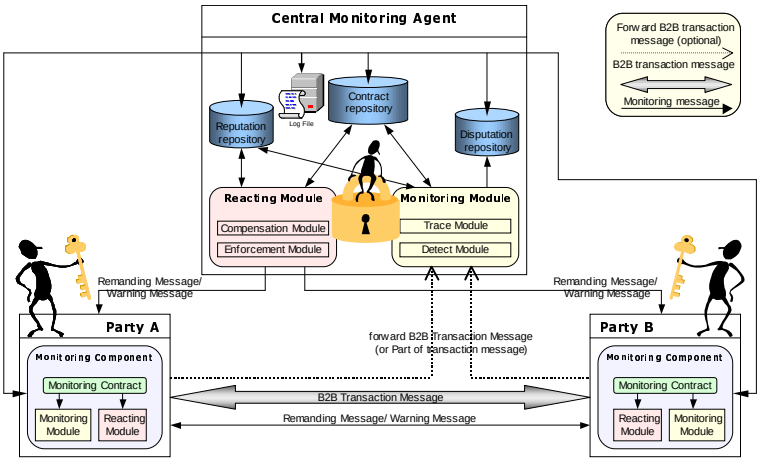


Figure 6.3: The architecture of contract fulfillment monitoring

At the execution of a contract, every party can forward all or a part of the transaction messages to the central monitoring agent; it is up to the business parties to decide how much they want to be monitored. Based on these messages, the central monitoring agent retrieves the particular contract from the contract repository, traces the process to identify which obligation remains on the party and the following actions that can be expected. Combining each partner's reputation with the current monitoring result, the central agent will send the relevant reminding or warning messages, use the enforcement mechanism to enforce conforming actions, identify the necessary compensating action based on the compensation clause in the contract if any violation happened, or record any unsolvable disputations in the disputation repository.

The monitoring and reactive modules of normal agents have the same function as those of the central monitoring agent. The normal agents' reactive and monitoring modules can communicate with those of the central monitoring agent to share

information, and they can connect to the central knowledge bases and the log file, so that they can monitor business processes when the agents did not opt for the central monitoring.

6.3 Summary

This chapter sketched a two-level monitoring framework of e-market for monitoring contract fulfillment. In this framework and in our monitorable contract, anomalous actions can be prevented, and deviating actions can be detected and compensated. In short, our monitorable e-contract can be kept in a contract repository, and the relevant peer partners have the appropriate contract that they need to fulfill. The monitoring module of the central monitoring party or the peer party can be implemented by the monitoring module in the monitoring mechanism which we mentioned in Chapter 5. The reactive module of the central monitoring party or the peer party is domain-dependent. In this chapter, we demonstrated how to develop a more reliable e-marketplace that behaves more similar to real-world marketplaces.

Chapter 7

Implementation and Evaluation

This chapter describes an implementation for monitorable contracts, explaining how a contract execution is expressed and how partners responsible for a contract violation are detected. Also, an efficiency evaluation and a complexity analysis of our implementation are presented.

We first define what is meant by an occurrence and demonstrate how occurrences can be used to represent a contract execution (Section 7.1). Next, we show how the maintaining guard algorithm and the pro-active detection algorithm can be represented in our prototype (Section 7.2). Then, we discuss how to detect the partners responsible for a contract violation (Section 7.3). Finally, an efficiency evaluation and a complexity analysis of implementation of monitorable contracts are provided (Section 7.4).

Our prototype is implemented in SWI-prolog [swi]. Prolog allows for efficient prototyping and is ideal for implementing the guard checking. The SWI-prolog environment offers multi-threading [swi], which is used to dynamically warn relevant partners which should perform the sequential actions in a contract execution.

7.1 Representing occurrences

In a workflow system, each action needs to occur in order. We treat an occurrence as being an instance of a specific action at a moment in time or over an interval in time. For instance, we could tread *phoneClaim, assignGarage, sendCar* and *repairCar* as occurrence actions. When an action occurs, it needs to be checked whether its pre-requisite actions have taken place. Using guards of contract constraints of a trying action checks whether this action could occur.

We view a contract execution as a set of action occurrences. To represent an action occurrence, first we express actions (Section 7.1.1), which relate with three static types and three dynamic types. Second, we provide an expression of various contract constraints (Section 7.1.2), and finally we present rules of guards (Section7.1.3).

7.1.1 Expressing actions

Each action instance has role-player acting in a subject role and an object role in the occurrence. For example, an action *repairCar* has a participant in the subject role: *garage* (a sender of action *repairCar*) and in the object role: *policyholder* (a receiver of action *repairCar*). Each action instance has a deadline, which is an interval time in our prototype. For example, the deadline of *repairCar* means that action *repairCar* should occur within 5 days after the garage receives the *agreeRepair* action. The three static types of an action thus are *the subject role or sender*, *the object role or receiver* and *the deadline*.

Each action instance also has three associated dynamic types: *absolute deadline, perform time* and *state*. In an execution process, the data of dynamic types will be updated in term of specific relationships between actions. In our prototype, the deadline of an action is a relative time. When the former action has occurred at a certain time, the *absolute deadline* of continuous actions can be calculated. *perform time* records the actual time at which the action is performed, which could be used as evidence in future attempts to find the responsible partners for a contract violation. *states* of an action indicate a variable process of the action among *not yet occurred*, *perhaps will occur*, *has occurred* and *will never occur*. For examples, if a is an action, the states of action a could change:

1. $\neg a \rightarrow \Diamond a$:from *not yet occurred* to *perhaps will occur*,
2. $\neg a \rightarrow \Box a$: from *not yet occurred* to *has occurred*,
3. $\Diamond a \rightarrow \Box a$: from *perhaps will occur* to *has occurred*,
4. $\Diamond a \rightarrow \Box \bar{a}$: from *perhaps will occur* to *will never occur*.

We have chosen a binary relation to structure data for each action. We use an example of action *repairCar* to explain how an action has been represented in our prototype. All the action *repairCar* information could have been written in terms of binary relations as follows:

```
sender(repairCar, garage).
receiver(repairCar, policyholder).
deadline(repairCar,5).
:-dynamic absolute_deadline/3.
absolute_deadline(repairCar,0,0).
:-dynamic perform_time/3.
perform_time(repairCar,0,0).
:-dynamic state/2.
state(repairCar,notyet).
```

The major merit of this structure is that it is easy to extend properties of an action. However, rules would then be expressed differently than a long-argument structure of an action, which could make implicit connections explicit.

7.1.2 Expressing contract constraints

Each contract constraint is expressed as a binary relational date structure as well. For instance, action *forwardInvoice*, action *returnClaimForm* and action *payRepairCost* belongs to a *joint initiation constraint* $((\bar{a} \vee \bar{b}) \wedge \bar{c} \vee a \cdot b \cdot c \vee b \cdot a \cdot c)$ in Chapter 4. In our prototype, this relationship is expressed as follows:

```
constraint2((forwardInvoice|returnClaimForm),payRepairCost).
```

Other contract constraints schemes in Chapter4 have similar data abstraction in our prototype. More details can be found in Appendix B.

7.1.3 Expressing guards

In our prototype, guards are represented as preconditions for a trying an action and postconditions after occurrence of an action in our implementation. To explain we look at the *join initiation contract constraint*: $(\overline{a}\vee\overline{b})\wedge\overline{c}\vee a\cdot b\cdot c\vee b\cdot a\cdot c$. When action a tries, the guard of the *join initiation contract constraints* $\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c}\vee a\cdot b\cdot c\vee b\cdot a\cdot c, a)$ is $\Diamond\overline{b}\wedge\Diamond\overline{c}\vee(\neg b\wedge\neg c\wedge\Diamond(b\cdot c))\vee(\Box b\wedge(\neg c\wedge\Diamond c))$, which could be regarded as two parts. The first case, when action a is trying and action b has not yet occurred, perhaps action b will occur before action c (i.e. $\neg b\wedge\neg c\wedge\Diamond(b\cdot c)$). In the second case, when action a is trying and action b has occurred, action c has not yet occurred right now and perhaps will occur (i.e. $\Box b\wedge(\neg c\wedge\Diamond c)$). A clear pattern can be seen, after action a has occurred, and action b did not yet occur, the partner who should perform action b will receive *reminding* information. If b did occur, a new thread is triggered, which will send a *warning* message to the partner who performs action c.

```
postcondition(A):- constraint2((A|B),C)  ->
(state(B,ST), ST == box ->
 thread_create(warning(C), ThreadID,[detached(true)]);
          reminding(B).
```

This rule generalizes other guards of the *join initiation contract constraints*. It is the same way that a rule postcondition(B) represents $\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c}\vee a\cdot b\cdot c\vee b\cdot a\cdot c, b)\equiv\Diamond\overline{a}\wedge\Diamond\overline{c}\vee(\Box a\wedge\neg c\wedge\Diamond c)\vee(\neg a\wedge\neg b\wedge\Diamond(a\cdot c))$ as follows:

```
postcondition(B):- constraint2((A|B),C)  ->
(state(A,ST), ST == box ->
 thread_create(warning(C), ThreadID,[detached(true)]);
          reminding(A).
```

When action c tries, the guard of the *join initiation contract constraint* ($\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c}\vee a\cdot b\cdot c\vee b\cdot a\cdot c, c)$) is $\Box a\wedge\Box b$, which could be regarded as a *precondition(C)*for action c, including four parts. The first part, action a has not yet occurred and action b has occurred, $\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c}\vee a\cdot b\cdot c\vee b\cdot a\cdot c, c)$ cannot be true, consequently action c cannot occur. The second part, action a has occurred and action b has not yet occurred, $\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c}\vee a\cdot b\cdot c\vee b\cdot a\cdot c, c)$ cannot be true either, since sequentially action c cannot occur. The third part, both action a and action b have not yet occurred, definitely $\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c}\vee a\cdot b\cdot c\vee b\cdot a\cdot c, c)$ cannot be true and action c cannot occur. The four part, when both action a and action b have occurred, $\mathsf{G}((\overline{a}\vee\overline{b})\wedge\overline{c}\vee a\cdot b\cdot c\vee b\cdot a\cdot c, c)$ becomes true and action c may occur. We define the rule *precondition(C)* as follows:

```
precondition(C):- constraint2((A|B),C)  ->
(((state(A,ST1), state(B,ST2),
          (ST1 == notyet| ST1 == diamond), ST2 == box)
           -> reminding(A), fail;!),
  ((state(A,ST1), state(B,ST2),ST1 == box,
          (ST2 == notyet| ST2 == diamond))
```

```
            -> reminding(B), fail;!),
((state(A,ST1), state(B,ST2),
            (ST1 == notyet | ST1 == diamond),
            (ST2 == notyet | ST2 == diamond)) ->
            reminding(A),reminding(B),!, fail;!),
((state(A,ST1), state(B,ST2),ST1 == box, ST2 == box,
            intime(C)) ->true)).
```

Above rules depicted guards of the *join initiation contract constraints*. Other contract constraints can be expressed in a similar way, where each guard calculation can be mapped to rules. For more detail see Appendix B.

7.2 Pro-active detection expression

In Chapter 4, we described maintain guard algorithm and pro-active detection algorithm respectively. It is clear that they are working together to realize a goal of proactive detection. In our prototype, the maintain guard algorithm is simply regarded to be all state changes of each actions. In this section, we explain how a pro-active detection using rules is implemented in our prototype.

After an action occurs, which actions can be expected to occur. Action occurrences refer to rules when reminding and warning modules are invoked. After an action occurs, all contract constraints related to this action will be reviewed. For example, in the *join initiation contract constraint* $((a \vee b) \wedge c \vee a \cdot b \cdot c \vee b \cdot a \cdot c)$, after action a has occurred, and action b has occurred as well, a reminding action c module is invoked, and a warning thread is invoked as well. In this thread, a warning function is triggered when action c does not occur 20 seconds before the deadline of action c. 20 seconds is just an implementation choice, and delay is allowed by the system. After action a has occurred, if action b has not yet occurred, a remind action b module is invoked. This process is illustrated as rule *postcondition(A)* as follows:

```
postcondition(A):- constraint2((A|B),C) -> (state(B,ST),
                    ST == box ->
                   (thread_create(warning(C),
                   ThreadID,[detached(true)]);
                   reminding(B).
```

In this section, we explain how algorithms of maintain guard and pro-active detection are implemented at our prototype. We will discuss how to find responsible partners after a contract violation in the next section.

7.3 Checking responsibility of contract violation

In multi-partner contractual environments, checking responsibility of a contract violation is inevitable. Advanced facilities for detection contract violations between multiple partners are therefore critical for a realistic e-market. In Chapter 5, we discussed how to detect responsible partners of a contract violation. In this section, we explain how to implement detection in our prototype. To find responsible partners for a contract violation we use commitments. We first explain how all

information of a commitment is expressed (section 7.3.1). Based upon this commitment data abstraction, rules of checking responsibility of a contract violation are defined in section 7.3.2.

7.3.1 Express commitments

Each commitment instance includes a subject partner who is called *commitment sender*, an object partner who is called *commitment receiver*, and all actions involved in a particular commitment. An action involved in a commitment has also another attribute to explain whether this action triggers, involves, or finishes this commitment. For example, all information of commitment *dailyService* can be written as follows:

```
c_sender(dailyService, lee_cs).
c_receiver(dailyService, agfil).
c_action(dailyService, forwardClaim, tr).
c_action(dailyService, contactGarage, in).
c_action(dailyService, sendRepairCost, in).
c_action(dailyService, agreeRepair, fi).
c_action(dailyService, repairCar, tr).
c_action(dailyService, sendInvoice, in).
c_action(dailyService, forwardInvoice, fi).
```

In this example, commitment *dailyService* includes actions *forwardClaim*, *contactGarage*, *sendRepairCost*, *agreeRepair*, *repairCar*, *sendInvoice* and *sendInvoice*. All actions have their attributers, "tr" for "trigger", "in" for "involve", and "fi" for "finish", which consists with **Definition** 4.2.2.1.

Actions *forwardClaim* and *repairCar* trigger commitment *dailyService*; actions *agreeRepair* and *forwardInvoice* finish commitment *dailyService*.

Orders between commitments are important when retrieving commitments and actions after a contract violation. Orders between commitments are also written in terms of binary relations as follows:

```
order_commitment(phoneService, repairService).
order_commitment(phoneService, dailyService).
order_commitment(phoneService, claimForm).
order_commitment(phoneService, inspectCar).
order_commitment(repairService, payRepairCost).
order_commitment(dailyService, payRepairCost).
order_commitment(inspectCar, payRepairCost).
order_commitment(claimForm, payRepairCost).
```

This example shows commitment *phoneService* is the first commitment and commitment *payRepairCost* is the last commitment at a contract execution of the car insurance case.

7.3.2 Rules of checking responsibility of a contract violation

After a contract violation, it is very important to detect who is/are responsible for this violation. In Chapter 5, we introduced our approach, in which the detecting process is based on retrieving all triggered commitments and actions involved in

all triggered commitments. In our prototype, the steps for detecting responsible partners are as follows:

i) detect all commitments which have triggered,

ii) retrieve all actions for each triggered commitment, check state of all actions and find all partners which did not perform actions that should have occurred,

iii) notify all partners of this violation.

In our prototype, the rule for detecting responsible partners is expressed as follows:

```
responsibility:-
  forall((state(A,never);state(A,diamond)),
   (forall(c_action(C,A,_),
     (write('Commitment_'), write(C),
        write('_is_violated!'),nl,
      sender(A,P),write(P),
      write('violate_his_promise.'),nl,nl)),
    forall((state(B,diamond),intime(B)),
     (sender(B,S), write(S),
        write('_pleases_stop_to_do_action_'),
        write(B),
        write(',_because_a_violation_is_found!_'), nl,
        state(A,never),    sender(A,S1),
        write(S1), write('_did_not_do_action_'),
        write(A), nl)))).
```

In this section, we explained a data abstraction of a contract and demonstrated how pro-active detection and contract violation detection are implemented in our prototype. We discuss the evaluation of our prototype in the next section.

7.4 Evaluation

This section provides a qualitative synopsis of the features of our approach. A complexity analysis in section 7.4.1 and a qualitative evaluation of our prototype's performance is presented in section 7.4.2.

7.4.1 Theoretical complexity analysis

Complexity analysis of the pro-active detection algorithm shows a polynomial worst-case time complexity of order $T(n(n-1))$ where n is the number of actions in the workflow process. Each action occurrence is based on a precondition and a post condition checking. Checking the precondition and postcondition depends on how many contract constraints in total in a contract. In the worst case, it is $n-1$ contract constraints in a contract execution. Finishing one workflow process requires that all actions have been performed. It is easy to show that a time complexity of a pro-active algorithm is $T(n(n-1))$.

Complexity analysis of the pro-active detection algorithm shows a lineal worst-case space complexity of order $O(3n)$ with only a reminding function, $O(4n)$ with a reminding function and a warning function where n is the number of actions.

7.4.2 Performance

The goal of our experiments was to validate that our approach is efficient, can be applied in practice, and scales to different contract size in long-running systems. Our prototype's performance can be compared with a pure action occurrence, a contract execution with reminding functions and a contract execution with reminding and warning functions.

There are some important things we have to point out before start to analyze our experiments. The prototype has several typical properties of prototypes such as a limited speed. However, this prolog prototype shows the worst case of our approach which actually works.

Methodology
For a contract, we generated three occurrence processes as follows:

1. ideal occurrence performance processes without any pro-active detective function,
2. occurrence performance processes with a reminding function,
3. occurrence performance processes with a reminding and warning function.

All experimental runs were generated with the same parameter. All experimental runs were repeated 1000 times. For all parameters see Appendix C.
Time complexity
In our experiment, we inserted the number of actions in a contract. The time of a execution contract taken to insert. The trend line overlaid on the data points in Figure 7.1 are consistent with the polynomial time complexity anticipated in secction 7.4.1.

Figure 7.1 and Table 7.4.2 show a performance time of a contract with 7 actions and 5 contract constraints at the first column, a contract with 16 actions and 14 contract constraints at the second column, a contract with 19 actions and 17 contract constraints at the third column.

For each contract, for example, in the third column of Figure 7.4.2, it is a contract with 7 actions and 5 contract constraints included. We compare three cases. The first case, all actions occur after each other without any delay. The total time of this contract execution is 1.02 seconds. The second case, all actions occur after receiving all reminding messages. Execution this contract costs 1.5 seconds. The third case, all actions occur after receiving all reminding and warning messages. It takes 2.2 second to fulfill this contract. There was not actual delay implemented in this test.

In this experiment, the time for the first case really short, which is not going to happen in a real business environment. For example, after a damaged car was delivered, the garage is normally not going to repair it immediately or can not finish to repair immediately anyway. We do not consider, how many time is needed for finishing certain actions. It is a reason why there are a big difference among the first case (1.02 seconds), the second case (1.5 seconds) and the third case (2.2 seconds) in the first contract (with 7 actions and 5 contract constraints). There are same differences in the second contract (with 16 actions and 14 contract constraints) in the second column of Talbe 7.4.2 and the third contract (with 19 actions and 17 contract constraints) in the thrid column of Table 7.4.2. The small value is gained

from the first case is an important factor to influence the efficiency of our approach. Anyway, our approach is not a such time-costing process indeed.

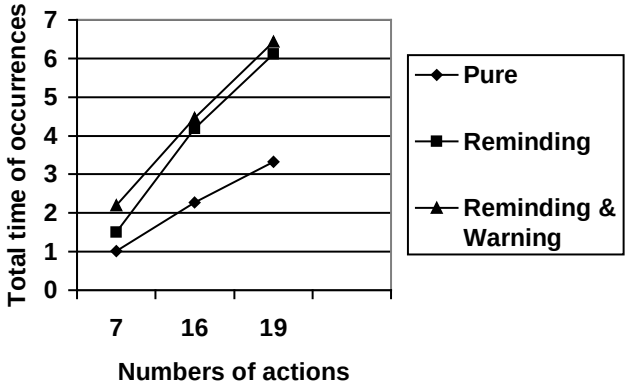


Figure 7.1: The performance time of different contracts

		Number of Actions		
		7	16	19
Total time, in second	Pure	1.02	2.27	3.33
	With reminding function	1.5	4.19	6.12
	With reminding and warning function	2.2	4.46	6.44

Table 7.1: Total time, in seconds, to insert

Space complexity

The space used by the recorded action states, absolute deadlines of actions, perform time of action occurrences. In a pro-active monitoring with a warning function, a thread number is recorded as well.

Figure 7.2 and Table 7.4.2 show a performance space of a contract with 7 actions and 5 contract constraints at the first column, a contract with 16 actions and 14 contract constraints at the second column, a contract with 19 actions and 17 contract constraints at the third column.

		Number of Actions		
		7	16	19
Total memory use	Pure	0	0	0
	With reminding function	21	48	57
	With reminding and warning function	28	64	76

Table 7.2: Total memory use, to insert

There are many programming languages, which can improve the memory use.

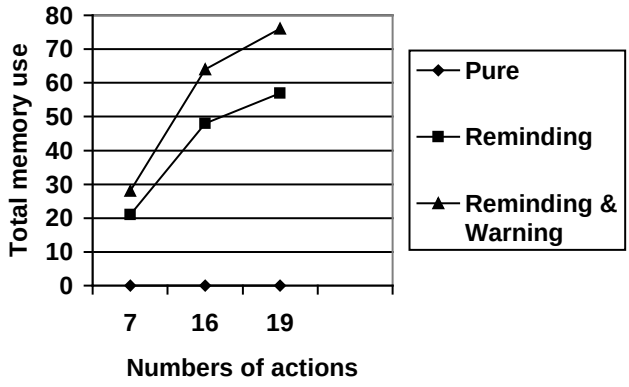


Figure 7.2: The performance space of different contracts

We only presented the worst case, which can absolutely be improved in another implementation.

7.5 Link to existing standards of systems

IBM WebSphere Application Server Enterprise Process Choreographer is a business-process engine that allows for the efficient execution of business processes. Business processes are sequences of activities that require coordination and business rules. In Figure 7.3, the process engine allows an application architecture that separates the description of the business logic (the flow logic) from the implementation of the business functions. The business logic is described as a process that consists of the steps in the process that need to be performed, their relationship to one another, and their ordering constraints.

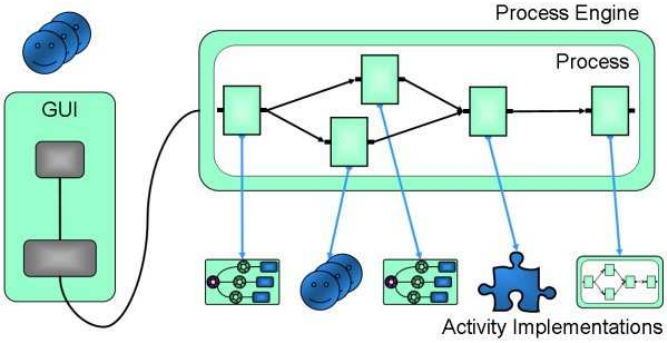


Figure 7.3: Structure of a business process-based application [KP02]

Our monitoring contract model also describes the business logic relationship as contract constraints. Naturally, monitoring module should include in the progress engine. Figure 7.4 shows where is our monitoring module in the WebSphere Process

Choreographer architecture. The Navigator component is the heart of the process engine. It manages the state transitions for all process instances, and the state transitions for all activities in those process instances. Our monitoring module could embed Nevigator which can handle contract constraints. To do this, monitorability of business processes could improve. The monitoring module consist of a commitment graph and a prolog engine. The commitment graph records multi-party contract information. The prolog engine will maintain guards of the contract constraints and trigger pro-monitoring information when it is necessary.

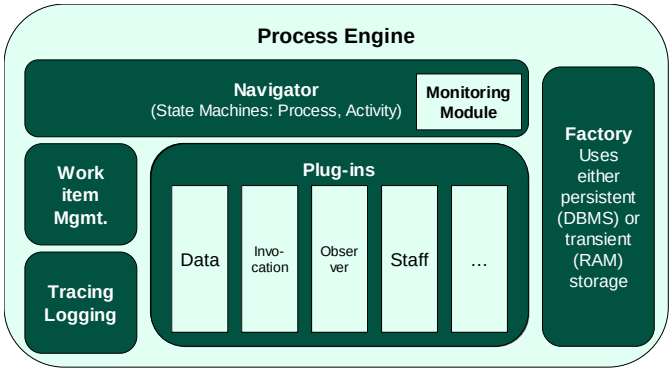


Figure 7.4: Structure of a business process-based application

Thus, besides concurrency, recoverability, heterogeneous and range of quality of service, properties of the business process-based applications also include monitorability.

7.6 Summary

Our experiments illustrate that our monitorable contract is efficient for monitoring contract fulfillment, showing that the approach is feasible for pro-active monitoring of multi-partner contract execution.

Chapter 8

Conclusions

This thesis explored the monitorability of multi-party contract. The research was motivated by the fact that all parties involved in e-business try to reduce the risk of business automation using e-contracts. Administrators of e-marketplaces want to provide better services for their participants. The automation of transactions in e-business and the looser bonds between business partners demand a higher level of control of the contract execution. Our research investigated in how far the content of a business contract between multiple partners can be transformed to monitoring rules that have a precise semantics and are executable by a program.

Unfortunately, administrators of existing e-marketplaces are currently able to offer only limited monitoring services because monitorable elements of a contract have not yet been formalized. The existing systems cannot give customers much information and are thus of only limited value in reducing the vulnerabilities of the involved parties.

Our monitoring theory defines precisely what a contract violation is and which business partner holds responsible for it. The monitoring mechanism not only detects actual violations but also imminent ones.

This chapter includes contributions of our research, answers of research questions and the future research in Sections 8.1, 8.2 and 8.3, respectively.

8.1 Contributions

The general contribution of this research is therefore that it provides a building block for improving e-business services using our monitorable contracts. This general contribution is support by the numbers of specific one, which are discussed later.

There are two kinds of research on business automations using e-contracts, which refers to two important concepts: "building a new business relationship using e-contract" and "contract provides a guarantee to contractual partners". Standards for establishment of business automation using the e-contract are provided in Section 2.3.1. They are only concerned with executing a business automation crossing all participating parties who are not belong to a single organization. This kinds of research belong to the first concept of the e-contract. There exist also other studies and standards for guaranteed Quality of Service (QoS) (in Section 1.2.1 and Section 2.3.1). They define the measurable services and quantitative performance

of business processes. This kinds of contracts are specifically used for monitoring the QoS, which represent the second concept of e-contract.

We have attempted to build a new bridge to connect the above-mentioned two research areas between the first concept and the second concept of the e-contract. In our research, contractual business executions cross the different organizations which satiate our first contract concept of "building a new business relationship". At the same time, during the business execution, our monitorable contract is able to trigger the system to send reminding and warning messages to relevant contractual parties when they are in a vulnerable situation. It fits our second contract concept of "contract provides a guarantee" to maintain certain levels' guarantee for the contract execution, that means it is not necessary to go lawsuit always. However, we do not quantify and evaluate services in our monitorable contract model.

To support monitoring contract during the contract fulfillment stage, there are three parts included in our research,

- *a monitorable contract model*, which is a formalization of the multi-party contract,
- *a dynamic monitoring mechanism*, which dynamically calculate the contract states at the contract fulfillment stage, and send the relevant reminding and warning messages.
- *a two-level framework* within which the monitorable contract model and the dynamic monitoring mechanism can run.

In the following sections, the features of the three parts are explained, respectively.

8.1.1 Features of the monitorable contract model

In Chapter 4, a monitorable contract model is presented. The monitorable contract model includes the actions, the commitments, the contract constraints, the guards of the contract constraints, and the commitment graph. We noted the following main features of the monitorable contract model:

(i) *A mapping from the paper contract*
For building our monitorable contract model, a paper contract is directly mapped into the trading process part of our contract model , which includes

- the *actions* are atoms of the contract.
- the *commitments* are essentially guarantees by one partner to another partner that some action sequences will occur.

(ii) *A temporal logic-based contract model*
Logic is a well-established technique to formalize problem areas and to describe solutions by means of calculus. Temporal logic is a logic of propositions whose truth and falsity may depend on time, a necessary facility for pro-active monitoring. Based on the temporal logic specifications in Chapter 3, the paper contract further leading to the second part of our contract model is logic relationships, which includes the contract constraints and the guards of the contract constraints.

- the *contract constraints* express restrictions on the occurrence order among the actions in a business process.

– the *guards of the contract constraints* regard to actions, which have occurred, which have not occurred but are expected to occur, and which should not occur in the future.

The validity of these conditions is tested during contract execution.

(iii) *A multi-party support contract model*
Little research has been done on multi-party contracts. There are however a great deal of business processes involved multi-parties. Our approach is that a multi-party contract may be broken down to a set of commitments, which are then specified in the trading process part of our monitorable contract model. This differs significantly from existing approaches, which decompose multi-party contracts into multiple bilateral contracts.

In order to specify the relationship between commitments, we presented the commitment graph in our monitorable contract model. One of our main concerns was how to find the party (or parties) responsible for a contract violation. The commitment graph maintains the relationship between commitments. It is used to trace back the commitments after a contract violation and located the partners who violated the commitments.

(iv) *Extensions to the contract monitoring method*
In Section 4.3.1, the four schemes are presented. The correctness of the guard functions G for four generic schemes have been proven. It should be noted that this set of schemes can be extended to cover more complex contract constraints. In such case, the guard computation presented must subsequently be extended as well.

Our monitorable contract model is thus a more flexible, expressive and general contract model than previous contract models for the contract monitoring. Moreover, our monitorable contract model also goes beyond a bilateral contract model and supports specification of multi-party contracts.

We see further achievements of our model that enhance its suitability for electronic contract execution. This research proposes an approach to formalize electronic contracts into a set of representations that enable automatic monitoring.

8.1.2 Features of the dynamic monitoring mechanism

According to our logic-based monitorable contract, we can calculate which actions can be expected during the execution of a contract. We use guards of the contract constraints, which are dynamically updated by using the *maintaining guards algorithm*. The *pro-active detection algorithm* uses the guards and the deadline of actions together to trigger the related reminding and warning messages. The *maintaining guards algorithm* and *pro-active detection algorithm* may be used to forecast the imminent contract violations by checking the state of the so-called guard expressions ahead of the formal deadline of an expected action.

We now turn to the features of the dynamic monitoring mechanism, which are summarized here:

(i) *A dynamic monitoring method*
During the contract execution time, the guards of the contract constraints are automatically updated using the *maintaining guards algorithm*, and the

related reminding and warning messages are triggered according to the deadlines and the guards of the relevant actions by using the *pro-active detection algorithm*.

(ii) *The method is easy to embed in existing e-business frameworks*
Our execution model is compatible with workflow engines for distributed execution of multi-partner contracts. The action trace, for example, can be seen as the log of a Petri Net machine, which controls the execution of the contract. However, we do not require such an execution engine as long as a sequence of actions is generated that can be monitored. Using a workflow engine would have the advantage that the checks for the guards can be integrated into the workflow at the right place.

(iii) *A pro-active-oriented monitoring method*
When contracts are automatically monitored, the likelihood of violation decreases (as partners can be alerted in advance of a real violation), and opportunities for compensating violations are created. Early detection can further contain the costs of a violation. The reduced costs would also diminish the changes of lawsuits. Instead, the failing partner can then commit itself to a compensation that creates value for all partners. Without automatic monitoring, the detection of compensation opportunities is simply too costly to justify complete monitoring.

The thesis concentrated on pro-active monitoring, which has been approached by using the detailed representation of monitorable multi-party contracts. In order to pro-actively monitor multi-party contracts during the running time of the business process, our dynamic monitoring mechanism continually updates the contract performance states by using the guard maintaining algorithm and the pro-active detection algorithm. The design proposed was verified through a prototype implementation in Prolog. The prototype validates implementability of our approach. Although in the worst case, all actions need to be reminded or warned, this is unlikely to happen in the real life. Our prototype shows that computational costs are low.

8.1.3 Features of the framework

Apart from our monitorable contract model and dynamic monitoring mechanism, also a *two-level monitoring framework* was sketched (in Section 6.1) of the e-market for the monitoring contract fulfillment. Under such a framework, the contract violations can be prevented, whereas actual violations can be detected and compensated.

The features of this framework are summarized as follows:

(i) *A framework for providing overall information*
We propose a two-level framework for our pro-active contract monitoring. The two-level framework includes a central monitoring level and a local monitoring level. The central monitoring level is important for multi-party business processes to collect overall monitoring information and to arbitrate in the resolution of conflicts.

(ii) *The ability to adapt to different e-market infrastructures*
Different infrastructures exist for the e-markets. E-market infrastructures can roughly be classified into central-controlled e-markets and self-regulated

e-markets (in Chapter 6). To adapt to different e-market infrastructures, the two-level framework is used. The local monitoring level is necessary for flexible adaptation to different e-market infrastructures. The local monitoring level makes it possible to keep monitorability in case the central monitoring level is in an invalid state or does not exist.

This two-level monitoring framework was also used to satisfy different monitoring requirements from the contractual parties in case some parties would rather not be monitored centrally.

Thus, under this framework, contract violations can be prevented, and actual violations can be detected or compensated in the self-chosen monitoring level.

This thesis has proposed a method for improving monitorability of e-contracts in general at the contract fulfillment stage. Thus, the monitorable contract model has been formalized, the dynamic monitoring mechanism has been provided, and the two-level framework within which our monitorable contract model could run has been presented. Further, the existing approaches to e-commerce applications and their deficiencies were described in Chapter 2. In the next section, we will answer our research questions which are introduced in Chpater 1.

8.2 Answers to research questions

Now we are in a position to answer our three research questions during the pro-active and reactive monitoring stages, which we posed in Section 1.2.3 of Chapter 1.

At the pro-active monitoring stage, there are two monitoring functions that should be carried out by our monitorable contract:

1. *Given the current state of contract execution, which actions are expected from a partner in the future?*
   To overview a whole business process, our approach uses a Petri Net; the current state of contract execution can thus be given. The contract constraints express restrictions on the order of occurrences of actions in a business process. After we know the current state of contract execution, the expected action from other partners can be calculated by using the guards of the contract constraints and the maintaining guard algorithm.

2. *Is a contract violation imminent (i.e. likely to happen within a short time)? Which partner must be reminded or warned to fulfill his/her obligation?*
   According to the current state of contract execution, we identify actions expected from other parties. Based on the contract constraints and the guards of the contract constraints, warning messages will be sent to the partner who needs to fulfill his/her obligation, and reminding messages will be sent to the potential 'victim' of the imminent violation by using the pro-active detecting algorithm.

At the reactive monitoring stage, we answer the following question.

1. *Which partner is (or partners are) responsible for a contract violation?*
   In our monitorable contract model, each action is specified by the sender and receiver of the action. For one action's failure, it is easy to find a responsible party. However, a contract violation can be caused by the mistakes of

many contractual parties. In our formal contract model, we thus specify a commitment graph which explains relationships between commitments and relationships between actions and commitments in a complex multi-party contract. The responsible partner(s) can thus be efficiently found using the commitment graph.

Hence, all questions during the contract fulfillment stage are answered by using our monitoring approach. The next section will discuss our future work after our current research.

8.3 Future research

This thesis raises a number of issues for future work in both practical and theoretical terms:

- To further prove the soundness and completeness of our propositional temporal logic (PTL).
- To extend the contract constraints, as was noted in Chapter 4, for specifying different business cases. Consequently, the guard computation presented should also to be extended.
- To further explore the area of quality safeguards in electronic contracts. Lack of trust between partners may be dealt with by introducing a trusted third party that can sub-divide actions into parts that are then irrevocable or provide monitoring services. An electronic contract can be analyzed prior to its execution in order to avoid incomplete commitment structures. Specifically, one may verify whether any violation of a contract constraint can be traced back to a commitment (i.e. a partner who is responsible for the violation) and change the contract accordingly.
- To run our monitorable contract model with ebXML, RosettaNet or other e-commerce platforms and to show an improvement of monitorability, and its technical viability.
- To extend the current monitorable contract model to the contract negotiation stage. The monitorable contract model is currently used only in the contract execution stage. However, the core part of the monitorable contract is represented by the contract constraints, which are not in absolute sequence; this thus provides a possibility to dynamically renew constraints for each result of the negotiation. It is possible to explore the new mechanism to extend the current monitorable contract model from only the contract fulfillment stage to the full contract life cycle.
- To integrate the monitorable contract model with subjective logic, which could improve flexibility when contractual partners have different reputations and abilities. Furthermore, the integration could provide pro-active and evidence-based contract monitoring.
- To integrate the monitorable contract model into the service level agreement that include quantitative measure about the contract execution. A repository could be involved to store relevant details about contract execution and then use quantitative queries on that repository.

In this thesis, we identified some critical requirements of contract monitoring at the fulfillment stage. We then formalized the monitorable contract model, developed the dynamic monitoring mechanism and introduced the two-level framework within which our monitorable contract model can run.

Two significant research issues of contract monitoring are explored in our research. First, we presented a pro-active monitoring concept, which not only detects actual contract violations but also imminent ones. Second, we formalized a multi-party contract by using the concept of commitments. This two characters of our research provided significant differences from the existing contract monitoring research.

The design proposed was verified through a prototype implementation in Prolog. Although in the worst case, all actions need to be reminded and warned, this is unlikely to happen in the real life. Our prototype shows that computation costs are low.

Finally, the contract monitoring at the contract fulfillment is just on of the research issues in e-contracting. Many other still await our investigation.

Bibliography

[AB00] A.S. Abrahams and J.M. Bacon. Event-centric business rules in e-commerce applications. In *Proceedings of Workshop on Best Practices in Business Rule Design and Implementation at the ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2000)*, 2000.

[AB01a] A.S. Abrahams and J.M. Bacon. Event-centric policy specification for e-commerce applications. In *Proceedings of Policy 2001, Workshop on Policies for Distributed Systems and Networks*, Bristol, UK., 2001.

[AB01b] A.S. Abrahams and J.M. Bacon. Occurrence-centric policy specification for e-commerce applications. In *Proceedings of Workshop on Formal Modelling for Electronic Commerce (FMEC 2001)*, Norway, Oslo, 2001.

[AB01c] A.S.Abrahams and J.M. Bacon. Representing and enforcing e-commerce contracts using occurrences. In *Proceedings of the 4th International Conference on Electronic Commerce Research (ICECR4)*, Dallas, Texas, 2001. Edwin L. Cox School of Business, Southern Methodist University.

[AB01d] A.S.Abrahams and J.M. Bacon. Representing and enforcing electronic commerce contracts over a wide range of platforms using occurrence stores. In *Proceedings of 4th CaberNet Plenary Workshop*, Pisa, Italy, 2001.

[AB02a] A.S. Abrahams and J.M. Bacon. The life and times of identified, situated, and conflicting norms. In *Proceedings of Sixth International Workshop on Deontic Logic in Computer Science*, London, UK, 2002. Imperial College.

[AB02b] A.S. Abrahams and J.M. Bacon. A software implementation of kimbrough´s disquotation theory for representing and enforcing electronic commerce contracts. *Group Decision and Negotiations Journal. Special Issue on Formal Modelling in Electronic Commerce*, 11(6):1–38, 2002.

[Abr02a] A. Abrahams. An asynchronous rule-based approach for business process automation using obligations. In *the 3rd ACM SIGPLAN Workshop on Rule-Based Programming*, Pittsburgh, USA, 2002.

[Abr02b] A.S. Abrahams. *Developing and Executing Electronic Commerce Applications with Occurrences*. PhD thesis, University of Cambridge Computer Laboratory, 2002.

[AF94] James F Allen and George Ferguson. Actions and events in interval temporal logic. *Journal of logic and computation*, 4(5):531–579, 1994.

[AG01] S. Angelov and P. Grefen. B2b econtract handling - a survey of projects, papers and standards. CTIT Technical Report 01-21, University of Twente, 2001.

[AG03] S. Angelov and P. Grefen. An analysis of b2b e-contracting domain: paradigms and required technology. Technical report, Research school for operations management and logistics of TUE, working paper WP102, 2003.

[All80] L.E. Allen. Language, law and logic: Plain legal drafting for the electronic age. In Niblett, editor, *Computer Science and Law, Cambridge University Press*. 1980.

[All82] L.E. Allen. Towards a normalised language to clarify the structure of legal discourse. In Martino, editor, *Deontic Logic, Computational Linguistics and Legal Information Systems*. 1982.

[All83] James F Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[AM90] M. Abadi and Z. Manna. Nonclausal deduction in first-order temporal logic. *Journal of the ACM*, 37(2):279–317, 1990.

[ASSR93] P. Attie, M. Singh, A. Sheth, and M. Rusinkiewicz. Specifying and enforcing intertask dependencies. In *Proceedings of the 19th Conference on Very Large Databases*, Los Altos (CA), 1993. Morgan Kaufman pubs, Dublin.

[Aus76] JL. Austin. *How to do things with words*. Oxford University Press, 2 edition, 1976.

[BHM+00] J. Bacon, A. Hombrecher, C. Ma, K. Moody, and W. Yao. Event storage and federation using odmg. In *Proceedings of 9th International Workshop on Persistent Object Systems, Design, Implementation and Use (POS9)*, volume 2135 of *Lecture Notes in Computer Science*, pages 265–281, Lillehammer, Norway, 2000. Springer-Verlag. Berlin, Germany.

[BLWW95] R. W. H. Bons, R. M. Lee, R. W. Wagenaar, and C. D. Wrigley. Modelling inter-organizational trade procedures using documentary petri nets. In *Proceedings of the Hawaii International Conference on System Sciences*, 1995.

[BMB+00] J. Bacon, K. Moody, J. Bates, R. Hayton, C. Ma, A. McNeil, O. Seidel, and M. Spiteri. Generic support for distributed applications. *IEEE Computer*, 33(3):68–77, 2000.

[BMY03] J. Bacon, K. Moody, and W. Yao. Access control and trust in the use of widely distributed services. *Software - Practice and Experience*, 33:375–394, 2003.

[BPM01] BPMI.org. Draft specification for the business process modeling language (bpml), draft 0.4, 8 2001.

[BPM03] Business process modeling language (bpml). `http://xml.coverpages.org/bpml.html`, 2003.

[CCK+02] Dickson K. W. Chiu, Shing-Chi Cheung, Kamalakar Karlapalem, Qing Li, and Sven Till. Workflow view driven cross-organizational interoperability in a web-service environment. In *Web Service, E-business and Semantic Web Workshop with CAiSE'02*, 2002.

[CCT02] Shing-Chi Cheung, Dickson K. W. Chiu, and Sven Till. A three-layer framework for cross-organizational e-contract enactment. In *Proceedings of Web Service, E-business and Semantic Web Workshop with CAiSE'02*, 2002.

[CCT03] Shing-Chi Cheung, Dickson K. W. Chiu, and Sven Till. Data-driven methodology to extending workflows to e-services over the internet. In *36th Hawaii International Conference on System Sciences*, 2003.

[CGK+02] Francisco Curbera, Yaron Goland, Johannes Klein, Frank Leymann, Dieter Roller, Satish Thatte, and Sanjiva Weerawarana. Business process execution language for web services, 2002. `http://www.ibm.com/developerworks/library/ws-bpel/`.

[Che80] B.F. Chellas. *Modal Logic: An Introduction*. Cambridge University Press, 1980.

[CKRa00] C.Dellarocas, M. Klein, and J.A. Rodriguez-aguilar. An exception-handling architecture for open electronic marketplaces of contract net software agents. In *the 2nd ACM Conference on Electronic Commerce*, 2000.

[CL90] Philip R Cohen and Hector J Levesque. Intention is choice with commitment. *Artificial intelligence*, 42(2-3):213–261, 1990.

[CL95] Philip R. Cohen and Hector J. Levesque. Communicative actions for artificial agents. In Victor Lesser and Les Gasser, editors, *Proceedings of the First International Conference on Multi-Agent Systems*, pages 65–72, San Francisco, CA, USA, 1995. The MIT Press: Cambridge, MA, USA.

[CNF01] G. Cugola, E. D. Nitto, and A. Fuggetta. The jedi event-based infrastructure and its application to the development of the opss wfms. *IEEE Transaction of Software Engineering (TSE)*, 27(9):827–850, 2001.

[CRW98] A. Carzaniga, D.S. Rosenblum, and A.L. Wolf. Design of a scalable event notification service: Interface and architecture. Technical Report CU-CS-863-98, Department of Computer Science, University of Colorado, 1998.

[CRW01] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.

[Dao98] F. Daoud. A business contracting model for tina architecture. *"Electronic Markets, International Journal of Electonic Markets*, 8(3):23–27, 1998.

[Das99] A. Daskalopulu. *Logic-Based Tools for the Analysis and Representation of Legal Contracts*. PhD thesis, Department of Computing, Imperial College, University of London, 1999.

[DDM01] A. Daskalopulu, T. Dimitrakos, and T.S.E. Maibaum. E-contract fulfilment and agents' attitudes. In *Proceedings ERCIM WG E-Commerce Workshop on The Role of Trust in e-Business*, 2001.

[DDN+98] A. Dan, D. Dias, T. Nguyen, M. Sachs, H. Shaikh, R. King, and S. Duri. The coyote project: Framework for multi-party e-commerce. In *Proceedings of the 7th Delos Workshop on Electonic Commerce*, Crete, Greece, 1998.

[DE95] Jorg Desel and Javier Esparza. *Free Choice Petri Nets*. Cambridge University Press, 1995.

[Del00] C. Dellarocas. Contractual agent societies: Negotiated shared context and social control in open multi-agent systems. In *Workshop on Norms and Institutions in Multi-Agent Systems, 4th International Conference on Multi-Agent Systems*, Barcelona, Spain, 2000.

[DKRR98] Hasan Davulcu, Michael Kifer, CR Ramakrishnan, and IV Ramakrishnan. Logic based modeling and analysis of workflows. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 25–33. ACM, 1998.

[DM01] A. Daskalopulu and T. S. E. Maibaum. Towards electronic contract performance. In *Proceedings of 12th Conference and Workshop on Database and Expert Systems Applications*, pages 771–777. IEEE C. S. Press, 2001.

[DND+01] A. Dan, T. N . Nguyen, D. M. Dias, F. N. Parr, R. Kearney, M. W. Sachs, T. C. Lau, and H. H. Shaikh. Business-to-business intergration with tpaml and a business-to-business protocol framework. *IBM Systems Journal*, 40(1), 2001.

[DP97a] A. Dan and F. Parr. The coyote approach for network centric business service applications. In *Proceedings of High Performance Transaction Processing (HPTS) Workshop*, Asilomar, CA, USA, 1997.

[DP97b] A. Dan and F. Parr. An object implementation of network centric business service applications (ncbas). In *Proceedings of OOPSLA Business Object Workshop*, Atlanta, GA, USA, 1997.

[DP99] A. Dan and F. Parr. Long running application models and cooperating monitors. In *Proceedings of High Performance Transaction Processing (HPTS) Workshop*, Asilomar, CA, USA, 1999.

[DTM02] A. Daskalopulu, T. Dimitrako T, and T. Maibaum. Evidence-based electronic contract performance monitoring. *INFORMS Journal of Group Decision and Negotiation*, Special Issue: formal Modeling of Electronic Commerce, 2002.

[Dub02] Jean-Jacques Dubray. A new model for ebxml bpss multi-party collaborations and web services choreography, 2002. `http://www.ebpml.org/ebpml.doc`.

[DWX01] Virginia Dignum, Hans Weigand, and Lai Xu. Agent societies: towards frameworks-based design. In *The Second International Workshop on Agent-Oriented Software Engineering (AOSE-2001), Lecture Notes in Computer Science*, volume 2222. Springer-Verlag, 2001.

[Eme90] E. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B. Elsevier Science Publishers, North-Holland, Amsterdam, 1990.

[Eng99] J. Engelfriet. *The Dynamics of Reasoning.* PhD thesis, Free University, Amsterdam, 1999.

[GA02] P. Grefen and S. Angelov. On t-, m-, p- and e-contracting. In *CAiSE Workshop on Web Services, e-Business, and the Semantic Web*, 2002.

[Gar97] A.v.d. L. Gardner. *An Artificial Intelligence Approach to Legal Reasoning.* MIT Press, 1997.

[GKP99] R. Gruber, B. Krishnamurthy, and E. Panagos. The architecture of the ready event notification service. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems Middleware Workshop*, 1999.

[GLA02] P. Grefen, H. Ludwig, and S. Angelov. A framework for e-services: A three-level approach towards process and data management. Technical Report CTIT Technical Report 02-07; IBM Research Report RC22378;, University of Twente, IBM Research Division, 2002.

[GSSS00] M. Greunz, B. Schopp, and K. Stanoevska-Slabeva. Supporting market transactions through xml contracting containers. In *the Sixth Americas Conference on Information Systems (AMCIS 2000)*, Long Beach, CA, 2000.

[GT98] Andreas Geppert and Dimitruis Tombros. Event-based distributed workflow execution with EVE. In Nigel Davies, Kerry Raymond, and Jochen Seitz, editors, *Middleware '98*. Springer-Verlag, 1998.

[GVA01] Paul Grefen, Jochem Vonk, and Peter Apers. Global transaction support for workflow management systems: from formal specification to practical implementation. *The VLDB Journal*, 2001.

[Hag96] S. Hagg. A sentinel approach to fault handling in multi-agent systems. In *the Second Australian Workshop on Distributed AI, in conjunction with the Fourth Pacific Rim International Conference on Artificial Intelligence (PRICAI'96)*, 1996.

[Hau02] Bob Haugen. Multi-party electronic business transactions, 2002. `http://www.supplychainlinks.com/MultiPartyBusinessTransactions.PDF`.

[Hin03] Annika Hinze. Efficient filtering of composite events. In *Proceedings of the 20th British National Database Conference*, 2003.

[Hoh13] W.N. Hohfeld. Some fundamental legal conceptions as applied in judicial reasoning. *Yale Law Journal*, 23, 1913.

[HS91] Joseph Y Halpern and Yoav Shoham. A propositional modal logic of time intervals. *Journal of the ACM (JACM)*, 38(4):935–962, 1991.

[JFJ+96] N. R. Jennings, P. Faratin, M. J. Johnson, P. O'Brien, and M. E. Wiegand. Using intelligent agents to manage business processes. In *1st Int. Conf. on The Practical Application of Intelligent Agents and Multi-Agent Technology*, 1996.

[Jös99] A. Jösang. An algebra for assessing trust in certification chains. In *Proceedings of NDSS'99, Network and Distributed Systems Security Symposium*, San Diego, 1999. The Internet Society.

[Jös01] A. Jösang. A logic for uncertain probabilities. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 9(3):279–311, 2001.

[KC00] Sanjeev Kumar and Philip R. Cohen. Towards a fault-tolerant multi-agent system architecture. In *the Fourth International Conference on Autonomous Agents (Agents 2000)*, pages 459–466, Barcelona, Spain, June 2000. ACM Press.

[KCK01] E. Kafeza, DKW. Chiu, and I. Kafeza. View-based contracts in an e-service cross-organizational workflow environment. In *the second International Workshop on Technologies for E-Service*, 2001.

[KD99] Mark Klein and Chrysanthos Dallarocas. Exception handling in agent systems. In Oren Etzioni, Jörg P. Müller, and Jeffrey M. Bradshaw, editors, *the Third International Conference on Autonomous Agents (Agents'99)*, pages 62–68, Seattle, WA, USA, 1999. ACM Press.

[KD01] Mark Klein and Chrysanthos Dellarocas. A knowledge-based methodology for designing robust electronic markets. Working Paper ROMA-WP-2001-02, ROMA, 2001.

[KGV99] M. Koetsier, P. Grefen, and Vonk. Contract model. Technical Report Deliverable D4b, Cross-Organisational/Workflow, Crossflow ESPRITE/28635, 1999.

[KGV00] M. Koetsier, P. Grefen, and J. Vonk. Contracts for cross-organizational workflow management. In *Proceedings of 1st International Conference on Electronic Commerce and Web Technologies*, pages 110–121, London, UK, 2000.

[KL03] A. Keller and H. Ludwig. The wsla framework ? specifying and monitoring service level agreements for web services. *Journal of Network and System Management, Special Issue on E-Business Management. Plenum Publishing Corporation*, 11(1), 2003.

[Kle91] Johannes Klein. Advanced rule driven transaction management. In *Compcon Spring'91. Digest of Papers*, pages 562–567. IEEE, 1991.

[Kle00] Mark Klein. Towards a systematic repository of knowledge about managing collaborative design conflicts. In *the International Conference on AI in Design*, 2000.

[KM93] S. O. Kimbrough and S. A. Moore. On obligation, time, and defeasibility in systems for electronic commerce. In *Proceedings of the 26th Hawaii International Conference on Systems Sciences*, pages 493–502, Kauai, Hawaii, 1993. IEEE Computer Society Press.

[KM97] S.O. Kimbrough and S.A. Moore. On automated message processing in electronic commerce and work support systems: Speech act theory and expressive felicity. *ACM Transactions on Information Systems*, 15(4):321–367, 1997. ACM Press. New York, NY.

[Kow79] R. Kowalski. *Logic fro Problem Solving.* Elsevier North Holland, 1979.

[KP02] M. Kloppmann and G. Pfau. Websphere application server enterprise process choreographer concepts and architecture. Technical report, IBM Corporation, 2002.

[Kri65] Saul A Kripke. Semantical analysis of intuitionistic logic i. 1965.

[Kro87] Fred Kroger. *Temporal Logic of Programs.* Springer-Verlag, 1987.

[KT98] Gal A. Kaminka and Milind Tambe. What is wrong with us? improving robustness through social diagnosis. In *the Fifteenth National Conference on Artificial Intelligence and Tenth Conference on Innovative Applications of Artificial Intelligence*, pages 97–104, Menlo Park, USA, 1998.

[Lee98a] R. M. Lee. A logic model for electronic contracting. *Decision Support Systems*, 4(1):27–44, 1998.

[Lee98b] R. M. Lee. Towards open electronic contracting. *International Journal of Electronic Market*, 8(3), 1998.

[Lee98c] Ron Lee. Towards open electronic contracting. *Electronic Markets*, 8(3), March 1998.

[Ley01] Frank Leymann. Web service flow language, 2001. `http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf`.

[LKD+03] H. Ludwig, A. Keller, A. Dan, R. King, and R. Franck. A service level agreement language for dynamic electronic services. *Electronic Commerce Research*, 3(1):43 – 59, 2003.

[LR95] R. M. Lee and Y. U. Ryu. Dx: A deontic expert system. *Journal of Management Information Systems*, 12(1):145–169, 1995.

[LS03] H. Ludwig and M. Stolze. Simple obligation and right model (sorm) - for the runtime management of electronic service contracts. In *Proceedings of the CAiSE'03 workshop: Web Services, e-Business, and the Semantic Web (WES): Foundations, Models, Architecture, Engineering and Applications (WES03), Lecture Notes of Computer Science*. Springer-Verlag, 2003.

[MAO96] Zoran Milosevic, David Arnold, and Luke O'Connor. Inter-enterprise contract architecture for open distributed systems: Security requirements. In *WET ICE'96 Workshop on Enterprise Security*, 1996.

[MBBR95] Z. Milosevic, A. Berry, A. Bond, and K. Raymond. Supporting business contracts in open distributed systems. In *2nd International Workshop on Services in Distributed and Networked Environments (SDNE'95)*, Whistler, Canada, 1995.

[Mey] B. Meyer. Building bug-free oo software: An introduction to design by contract. `http://www.eiffel.com/doc/manuals/technology/contract/`.

[Mey97] B. Meyer. *Object-Oriented Software Construction*. Prentice Hall, 2 edition, 1997.

[Mil95] Z. Milosevic. *Enterprise Aspects of Open Distributed Systems*. PhD thesis, Department of Computer Science, University of Queensland, 1995.

[MLA+02] B. Mehta, M. Levy, G. M. Tony Andrews, B. Beckman, J. Klein, and A. Mital. Biztalk service 2000 business process orchestration. In *International Conference on Data Engineering (ICDE'02)*, 2002.

[Moo00] S.A. Moore. Kqml and flbc: Contrasting agent communication languages. *International Journal of Electronic Commerce*, 5(1), 2000.

[MP92] Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems*. Springer-Verlag, 1992.

[MSBG01] O. Marin, P. Sens, J. Briot, and Z. Guessoum. Towards adaptive fault tolerance for distributed multi-agent systems. In *proc. ERSADS'2001*, Bertinoro, Italy, 2001.

[MsDP02] Z. Milosevic, A. Jøsang, T. Dimitrakos, and M.A. Patton. Discretionary enforcement of electronic contracts. In *The 6th IEEE internional Enterprise Distributed Object Computing ConferenceEDOC'2002*. IEEE Comp. Soc. Press, 2002.

[MSS97] M. Mansouri-Samani and M. Sloman. Gem: A generalized event monitoring language for distributed systems. *Distributed Systems Engineering Journal*, (4):96–108, 1997.

[MW93a] J-J. Ch. Meyer and R. J. Wieringa. Deontic logic: A concise overview. In Meyer and Wieringa, editors, *Deontic Logic in Computer Science: Normative Systems Specification*. John Wiley & Sons, 1993.

[MW93b] J.-J Ch. Meyer and R.J. Wieringa. *Deonic Logic in Computer Science: Normative Systems Specification*, chapter Deontic Logic: A Concise Overview. John Wiley & Sons, 1993.

[NSJ98] T. J. Norman, C. Sierra, and N. R. Jennings. Rights and commitments in multi-agent agreements. In *Proceedings of the 3rd International Conference on Multi-Agent Systems (ICMAS-98)*, Paris, France, 1998.

[OAS02] Oasis ebxml collaboration-protocol profile and agreement specification version 2.0. Organization for the Advancement of Structured Information Standards, 2002.

[oJC95] Department of Justice Canada. A survey of legal issues relating to the security of electronic information. Technical report, Department of Justice, 1995.

[Par96] H.Van Dyke Parunak. Visualizing agent conversations: Using enhanced dooley graphs for agent design. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96)*, 1996.

[Pau87] Lawrence C. Paulson. *Logic and Computation*. Cambridge University Press, 1987.

[PFL+00] João Pereira, Françoise Fabret, François Llirbat, Radu Preotiuc-Pietro, Kenneth A. Ross, and Dennis Shasha. Publish/subscribe on the Web at extreme speed. In Amr El Abbadi, Michael L. Brodie, Sharma Chakravarthy, Umeshwar Dayal, Nabil Kamel, Gunter Schlageter, and Kyu-Young Whang, editors, *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases*, pages 627–630, Cairo, Egypt, 2000. Morgan Kaufmann Publishers.

[Pnu85] A. Pnueli. Linear and branching structures in the semantics and logics of reactive systems. In *Proceedings of 12th International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science, pages 15–32. Springer-Verlag, 1985.

[Pri57] Arthur N Prior. *Time and modality*. OUP Oxford, 1957.

[Pro92] ALDUS Project. The aldus project: Artifical legal draftsman for use in sales. ESPRIT Commission, 1992.

[Pro99a] CrossFlow Project. Insurance requirements. Technical Report CrossFlow deliverable: D1.b, CrossFlow consortium, 1999.

[Pro99b] CrossFlow Project. Insurance scenario description. Technical Report CrossFlow deliverable: D1.a, CrossFlow consortium, 1999.

[Rei92] Wolfgang Reisig. *A Primer in Petri Net Design*. Springer-Verlag, 1992.

[RGWC99] Daniel M Reeves, Benjamin N Grosof, Michael P Wellman, and Hoi Y Chan. Toward a declarative language for negotiating executable contracts. In *the AAAI-99 Workshop on Artificial Intelligence in Electronic Commerce*, Menlo Park, CA, USA, 1999.

[RSSS99] A. Runge, B. Schopp, and K. Stanoevska-Slabeva. The management of business transactions through electronic contracts. In *Proceedings of the 10th International Workshop on Database and Expert Systems Applications (DEXA'99)*, Florence, 1999.

[RU71] N. Rescher and A. Urquhart. Temporal logic. *Springer-Verlag*, 1971.

[Ser91] M.J. Sergor. The representing legislation as logic programs. In Hayes, Michie, and Richards, editors, *Knowledge-Based Systems and Legal Applications*. Academic Press, 1991.

[Ser01] M. Sergot. A computational theory of normative positions. *ACM Transactions on Computational Logic*, 2(4):581–622, 2001.

[Sha02] Robert Shapiro. A comparison of xpdl, bpml, and bpel4ws. ebPML.org, 2002.

[Sin97] Munindar P. Singh. A customizable coordination service for autonomous agents. In *International Workshop on Agent Theories, Architectures, and Languages (ATAL)*, July 1997.

[SSC+01] A. Seaborne, E. Stammers, F. Casati, G. Piccinelli, and M. Shan. A framework for business composition. Position papers, the world wide web consortium (W3c workshop), 2001.

[swi] Swi-prolog. `http://www.swi-prolog.org`.

[Tha01] Satish Thatte. Xlang web services for business process design. `http://www.gotdotnet.com/team/xml\_wsspecs/xlang-c/default.htm`, 2001.

[TT99] Y-H. Tan and W. Thoen. A logical model of directed obligations and permissions to support electronic contracting in electronic commerce. *International Journal of Electronic Commerce*, 3(2):87–104, 1999.

[UNC87] UNCID. Uniform rules of conduct for interchange of trade data by teletransmission. `http://www.unece.org/trade/untdid/texts/d240\_d.htm`, 1987.

[V+90] Yde Venema et al. Expressiveness and completeness of an interval tense logic. *Notre Dame Journal of Formal Logic*, 31(4):529–547, 1990.

[Var01] Moshe Y. Vardi. Branching vs. linear time: Final showdown. In *Proceedings of 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 143–157, 2001.

[vdADtHW02] W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, and P. Wohed. Pattern-based analysis of bpml (and wsci). Technical Report QUT Technical report, FIT-TR-2002-05,, Queensland University of Technology, Brisbane, Australia, 2002.

[vG01] R.J. van Glabbeek. The linear time – branching time spectrum (1); the semantics of concrete, sequential processes. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, pages 3–99. Elsevier, 2001.

[VS99] Mahadevan Venkatraman and Munindar P. Singh. Verifying compliance with commitment protocols: Enabling open web-based multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 2(3), 1999.

[vW51] G.H. von Wright. Deontic logic. *Mind*, 1951.

[WC96] Jennifer Widom and Stefano Ceri. *Active Database Systems : Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann, San Francisco, California, 1996.

[WDV97] H. Weigand, F. Dignum, and E. Verharen. Dynamic business models as a basis for interoperable transaction design. *Information Systems*, 1997.

[WfM02] Wfmc announces release of workflow standard xml process definition language (xpdl 1.0), 2002. `http://www.wfmc.org/standards/docs/TC-1025\_10\_xpdl\_102502.pdf`.

[Wie98] M.J. Wieringa. *Norms, Logics and Information Systems*, chapter Normative Positions, pages 289–308. IOS Press, 1998.

[WJ94] Michael Wooldridge and Nicholas R Jennings. Agent theories, architectures, and languages: a survey. In *Intelligent agents*, pages 1–39. Springer, 1994.

[Woo92] Michael John Wooldridge. *The logical modelling of computational multi-agent systems*. PhD thesis, Citeseer, 1992.

[Wri65] G.H. Von Wrigh. *An Next*. Acta Philosophica Fennica, Fasc XVIII, North-Holland, 1965.

[Wri68] G.H. Von Wright. An essay in deontic logic and the general theory of action. *Acta Philosophica Fennica*, Fasc. XXI, 1968.

[WW99] S.M. H. Wallman and K. M. H. Wallman. Law and electronic commerce: The next frontier. *Telecommunications Reports International Journal*, 3(1), Winter 1999.

[WW01] Jane K. Winn and Benjamin Wright. *Law of Electronic Commerce, 4th Edition*. Aspen Publishers, 2001.

[WX01] H. Weigand and L. Xu. Contracts in e-commerce. In *9th IFIP 2.6 Working Conference on Database Semantic Issues in E-Commerce Systems (DS-9)*, 2001.

[XJ03] Lai Xu and Manfred A. Jeusfeld. Pro-active monitoring of electronic contracts. In *The 15th Conference On Advanced Information Systems Engineering in Lecture Notes of Computer Science*, volume 2681, pages 584–600. Springer-Verlag, 2003.

[Xu02a] Lai Xu. Agent-based monitorable contract. Research paper, Tilburg University, 2002.

[Xu02b] Lai Xu. Car insurance case. Research paper, Tilburg University, 2002.

[Xu03a] Lai Xu. A framework for e-markets: Monitoring contract fulfillment. In *the CAiSE'03 workshop: Web Services, e-Business, and the Semantic Web (WES): Foundations, Models, Architecture, Engineering and Applications (WES03),in Lecture Notes of Computer Science*. Springer-Verlag, 2003.

[Xu03b] Lai Xu. Monitorable electronic contract. In *The 2003 IEEE Conference on E-Commerce (CEC'03)*. IEEE Computer Society Press, 2003.

[Yos97] H. Yoshino. Leagal expert project. *Journal of Advanced Computational Intelligence*, 1(2):83–85, 1997.

[Yos98] H. Yoshino. Logical structure of contract law system for constructing a knowledge base of the united nations convention on contracts for the international sale of goods. *Journal of Advanced Computational Intelligence*, 2(1):2–11, 1998.

Appendix A

Car Insurance Case

We will use a standard multi-partner car insurance case [Pro99a] to explain our approach and to show that in a multi-partner contract it is more important and more difficult to find the responsible partner for a contract violation than in a bilateral contract.

This case outlines the manner in which a car damage claim is handled by an insurance company (AGFIL). The contract parties work together to provide a service level which facilitates efficient claim settlement. The parties involved are called Europ Assist, Lee Consulting Services, Garages and Assessors. Europ Assist offers a 24-hour emergency call answering service to policyholders. Lee C.S. coordinates and manages the operation of the emergency service on a day-to-day level on behalf of AGFIL. Garages are responsible for car repair. Assessors conduct the physical inspections of damaged vehicles and agree upon repair figures with the garages.

The general process of a car insurance case is described as follows: the policyholder phones Euro Assist using a toll-free phone number to notify a new claim. Euro Assist will register the information, suggest an appropriate garage, and notify AGFIL, which will check whether the policy is valid and covers this claim. After AGFIL receives this claim, AGFIL sends the claim details to Lee C.S. AGFIL will send a letter to the policyholder for a completed claim form. Lee C.S. will agree upon repair costs if an assessor is not required for small damages; otherwise, an assessor will be assigned. The assessor will check the damaged vehicle and agree upon repair costs with the garage. After receiving an agreement for repairing the car from Lee C.S., the garage will then commence repairs. After finishing repairs, the garage will issue an invoice to Lee C.S., which will check the invoice against the original estimate. Lee C.S. returns all invoices to AGFIL, which processes the payment. In the whole process, if the claim is found invalid, all contractual parties will be contacted and the process will be stopped.

The case study shows a rather complex workflow between multiple partners. In particular, the process can fail whenever one partner does not perform certain actions and such failure can have cascading effects on actions of other partners. Hence, tracing back failures to the originating partners is not trivial, and the failure might be uncovered by a contract clause.

The business process is presented in Figure A.1.

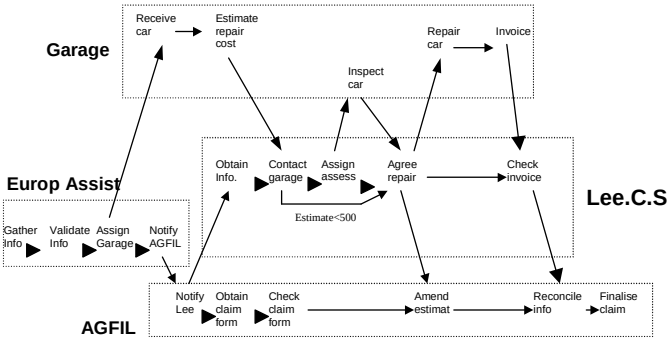


Figure A.1: The process diagram [Pro99b]

A.0.1 Overview of all parties

In this section, an overview of participating parties is presented in Figure A.2.

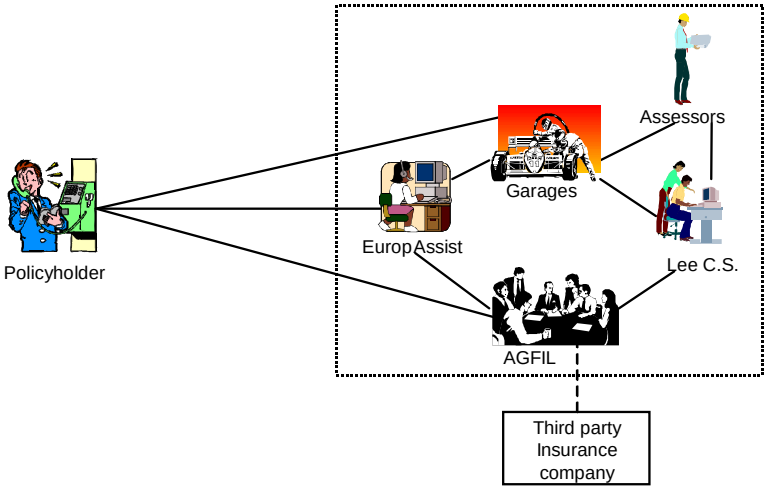


Figure A.2: Overview of all parties

All participating parties are specifies as following,

AGFIL:	AGFIL is the insurance company in the scenario, responsible for underwriting the motor policy and covering losses incurred. AGFIL holds ultimate control in deciding whether the claim is valid and payment will be made.
Europ Assist:	Europ Assist records the initial advice over the telephone and control is limited to minor evaluation of the loss incurred and to encourage the use of approved repairers/garages.
Lee C.S.:	Lee Consulting Services co-ordinates and manages the operation of the emergency service on a day-to-day level on behalf of AGFIL. Lee C.S. controls whether the vehicle requires assessment (only if the repair estimate exceeds a certain amount), and which Assessor/Adjuster is appointed to assess the damage. Lee also controls how quickly garages will receive payment, as all invoices are presented to AGFIL via Lee's. This element of control also extends to ensuring that repair figures are in line with industry norms. Lee's also prepares monthly accounts and reports.
Garages:	Approved repairers/garages provide the repair service and courtesy cars (if required). This is the most important element of the process from the policyholder's viewpoint, and these approved repairers control the success of the scheme. It is vital that the repair work is carried out in a timely and cost-efficient manner.

A.0.2 Contracts in the car insurance case

In this section, the contracts between each party are the following tables:

AGFIL	**Policyholders**
• A motor accident. – Recovered from the scene of the emergency. – Onward travel or overnight accommodation. – A free courtesy car for five days. • A motor loss through theft or fire. (this item is not addressed here). – A courtesy car for 14 days.	• Pay the agreed amount per month.

Table A.1: Outline of a contract between AGFIL and policyholders

AGFIL	**Europ Assist**
• Pays for Europ Assist's service.	• Receives phone calls from policyholders and collects and verifies the information. – Insurance company. – The caller's name, address, and phone number. – Claim Description. – Type of coverage. – Registration Number, Vehicle maker and model. – Policy Number. – Date and Time of Notification Loss. – Data of incident. – Injury to Driver, Passengers, Third Party? – Damage: Minor, Moderate or Severe – Vehicle Immobile? – Towing required? – Car rental required? – Company replacement car providing. • Assigns an approved garage for the policyholder. • Notifies AGFIL about this claim.

Table A.2: Outline of a contract between AGFIL and Europ Assist

AGFIL	**Lee Consulting Services**
• Pays for Lee C.S.'s service. • Notifies Lee C.S. about claim.	• Deals with the day-to-day business with garages. • Arranges agreed-upon repair. – If the repair cost is under a certain amount, repairs will be agreed upon over the phone. – If the repair cost is over a certain amount, the Lee C.S. will appoint an assessor to investigate. Lee C.S. has total authority to agree upon the repair figures on behalf of AGFIL. • Sends repair cost to AGFIL. • Checks invoices.

Table A.3: Outline of a contract between AGFIL and Lee Consulting Services

AGFIL	**Garage**
• Pays for Garage's service.	• Phones Lee C.S. when a vehicle is presented for repair. • Estimated repairs. • If an Assessor is appointed, discussions will take place and repair cost will be agreed upon. • Repairs vehicle. • Forwards Invoices. • Offers discount repair cost.

Table A.4: Outline of a contract between AGFIL and Garage

AGFIL	**Assessor**
• Pays for the assessor's service.	• Physically inspects the vehicle. • Agrees upon the costs of the repairs with the garage. • Forwards the repair cost to Lee C.S.

Table A.5: Outline of a contract between AGFIL and Assessor

Appendix B

Codes

```
/* ------------- Actions ------------*/
sender(phoneClaim, policyholder).
sender(receiveInfo, policyholder).
sender(assignGarage, euro_assist).
sender(sendCar,policyholder).
sender(estimateRepairCost, garage).
sender(notifyClaim, euro_assist).
sender(forwardClaim, agfil).
sender(contactGarage,lee_cs).
sender(sendClaimForm,agfil).
sender(returnClaimForm,policyholder).
sender(sendRepairCost,garage).
sender(assignAssessor,lee_cs).
sender(inspectCar,assessor).
sender(sendNewRepairCost,assersor).
sender(agreeRepair, lee_cs).
sender(repairCar,garage).
sender(sendInvoice,garage).
sender(forwardInvoice, lee_cs).
sender(payRepairCost,agfil).


receiver(phoneClaim,euro_assist).
receiver(receiveInfo, euro_assist).
receiver(assignGarage, policyholder).
receiver(sendCar,garage).
receiver(estimateRepairCost, policyholder).
receiver(notifyClaim, agfil).
receiver(forwardClaim, lee_cs).
receiver(contactGarage, garage).
receiver(sendClaimForm,policyholder).
receiver(returnClaimForm,agfil).
receiver(sendRepairCost, lee_cs).
receiver(assignAssessor,assessor).
```

```
receiver(inspectCar,lee_cs).
receiver(sendNewRepairCost,lee_cs).
receiver(agreeRepair, garage).
receiver(repairCar,policyholder).
receiver(sendInvoice, lee_cs).
receiver(forwardInvoice, agfil).
receiver(payRepairCost,garage).


deadline(phoneClaim,0).
deadline(receiveInfo,0).
deadline(assignGarage,0).
deadline(sendCar,1).
deadline(estimateRepairCost, 2).
deadline(notifyClaim, 1).
deadline(forwardClaim, 1).
deadline(contactGarage,1).
deadline(sendClaimForm,2).
deadline(returnClaimForm,7).
deadline(sendRepairCost,1).
deadline(assignAssessor,1).
deadline(inspectCar,1).
deadline(sendNewRepairCost,3).
deadline(agreeRepair,3).
deadline(repairCar,5).
deadline(sendInvoice,10).
deadline(forwardInvoice, 6).
deadline(payRepairCost,30).


/*----------------------Commitment----------------------*/
/* C_phoneService: A_phoneCalim, A_receiveInfo,
    A_assignGarage, A_notifyClaim*/
/* C_repairService: A_sendCar, A_estimateRepariCost,
    A_agreeRepair, A_repairCar */
/* C_claimForm: A_notifyClaim, A_sendClaimFrom,
    A_returnClaimFrom */
/* C_dailyService: A_forwardClaim, A_contactGarage,
    A_sendRepairCost, A_agreeRepair,
    A_repairCar ,A_sendInvoice, A_forwardInvoice*/
/* C_payRepairCost: A_forwardInvoice, A_returnClaimFrom,
    A_payRepairCost*/
/*------------------------------------------------------*/

c_sender(phoneService,euro_assist).
c_sender(repairService,garage).
c_sender(claimForm,agfil).
c_sender(dailyService,lee_cs).
c_sender(paymentSeriver,agfil).
```

```
c_receiver(phoneService, policyholder).
c_receiver(repairService, policyholder).
c_receiver(claimForm, policyholder).
c_receiver(dailyService, agfil).
c_receiver(paymentService, garage).

c_action(phoneService, phoneClaim, tr).
c_action(phoneService, receiveInfo, in).
c_action(phoneService, assignGarage, fi).
c_action(phoneService, notifyClaim, fi).

c_action(repairService, sendCar, tr).
c_action(repairService, estimateRepairCost, fi).
c_action(repairService, agreeRepair, tr).
c_action(repairService, repairCar, fi).

c_action(claimFrom, notifyClaim, tr).
c_action(claimFrom, sendClaimForm, in).
c_action(claimFrom, returnClaimForm, fi).

c_action(dailyService, forwardClaim, tr).
c_action(dailyService, contactGarage, in).
c_action(dailyService, sendRepairCost, in).
c_action(dailyService, agreeRepair, fi).
c_action(dailyService, repairCar, tr).
c_action(dailyService, sendInvoice, in).
c_action(dailyService, forwardInvoice, fi).

c_action(inspectCarService, assignAssessor, tr).
c_action(inspectCarService, inspectCar, in).
c_action(inspectCarService, sendNewRepairCost, fi).

c_action(paymentService, forwardInvoice, tr).
c_action(paymentService, returnClaimForm, tr).
c_action(paymentService, payRepairCost, fi).


/*----------------Constraints----------------*/

constraint(phoneClaim, receiveInfo).
constraint(receiveInfo, (assignGarage|notifyClaim)).
constraint(assignGarage, sendCar).
constraint(sendCar, estimateRepairCost).
constraint(receiveInfo, notifyClaim).
constraint(notifyClaim, (forwardClaim|sendClaimForm)).
constraint(forwardClaim, contactGarage).
constraint(sendClaimForm, returnClaimForm).
constraint((estimateRepairCost|contactGarage),
```

```
            sendRepairCost).
constraint(sendRepairCost,assignAssessor).
constraint(assignAssessor, inspectCar).
constraint(inspectCar, sendNewRepairCost).
constraint(sendNewRepairCost, agreeRepair).
constraint(agreeRepair, repairCar).
constraint(repairCar,sendInvoice).
constraint(sendInvoice, forwardInvoice).
constraint((forwardInvoice|returnClaimForm),payRepairCost).


init:-
        dynamic(absolute_deadline/3),
        dynamic(perform_time/3),
        dynamic(state/2),
        dynamic(threadID/2).

set:-
  assert(absolute_deadline(phoneClaim,start,start)),
  assert(absolute_deadline(receiveInfo,start,start)),
  assert(absolute_deadline(assignGarage,start,start)),
  assert(absolute_deadline(sendCar,start,start)),
  assert(absolute_deadline(estimateRepairCost, start,start)),
  assert(absolute_deadline(notifyClaim, start,start)),
  assert(absolute_deadline(forwardClaim, start,start)),
  assert(absolute_deadline(contactGarage,start,start)),
  assert(absolute_deadline(sendClaimForm,start,start)),
  assert(absolute_deadline(returnClaimForm,start,start)),
  assert(absolute_deadline(sendRepairCost,start,start)),
  assert(absolute_deadline(assignAssessor,start,start)),
  assert(absolute_deadline(inspectCar,start,start)),
  assert(absolute_deadline(sendNewRepairCost,start,start)),
  assert(absolute_deadline(agreeRepair,start,start)),
  assert(absolute_deadline(repairCar,start,start)),
  assert(absolute_deadline(sendInvoice,start,start)),
  assert(absolute_deadline(forwardInvoice, start,start)),
  assert(absolute_deadline(payRepairCost,start,start)),
  assert(perform_time(phoneClaim,0,0)),
  assert(perform_time(receiveInfo,0,0)),
  assert(perform_time(assignGarage,0,0)),
  assert(perform_time(sendCar,0,0)),
  assert(perform_time(estimateRepairCost, 0,0)),
  assert(perform_time(notifyClaim, 0,0)),
  assert(perform_time(forwardClaim, 0,0)),
  assert(perform_time(contactGarage,0,0)),
  assert(perform_time(sendClaimForm,0,0)),
  assert(perform_time(returnClaimForm,0,0)),
  assert(perform_time(sendRepairCost,0,0)),
  assert(perform_time(assignAssessor,0,0)),
```

```
  assert(perform_time(inspectCar,0,0)),
  assert(perform_time(sendNewRepairCost,0,0)),
  assert(perform_time(agreeRepair,0,0)),
  assert(perform_time(repairCar,0,0)),
  assert(perform_time(sendInvoice,0,0)),
  assert(perform_time(forwardInvoice, 0,0)),
  assert(perform_time(payRepairCost,0,0)),
  assert(state(phoneClaim,notyet)),
  assert(state(receiveInfo,notyet)),
  assert(state(assignGarage,notyet)),
  assert(state(sendCar,notyet)),
  assert(state(estimateRepairCost, notyet)),
  assert(state(notifyClaim, notyet)),
  assert(state(forwardClaim, notyet)),
  assert(state(contactGarage,notyet)),
  assert(state(sendClaimForm,notyet)),
  assert(state(returnClaimForm,notyet)),
  assert(state(sendRepairCost,notyet)),
  assert(state(assignAssessor,notyet)),
  assert(state(inspectCar,notyet)),
  assert(state(sendNewRepairCost,notyet)),
  assert(state(agreeRepair,notyet)),
  assert(state(repairCar,notyet)),
  assert(state(sendInvoice,notyet)),
  assert(state(forwardInvoice, notyet)),
  assert(state(payRepairCost,notyet)),
  assert(threadID(receiveInfo,0)),
  assert(threadID(assignGarage,0)),
  assert(threadID(sendCar,0)),
  assert(threadID(estimateRepairCost, 0)),
  assert(threadID(notifyClaim, 0)),
  assert(threadID(forwardClaim, 0)),
  assert(threadID(contactGarage,0)),
  assert(threadID(sendClaimForm,0)),
  assert(threadID(returnClaimForm,0)),
  assert(threadID(sendRepairCost,0)),
  assert(threadID(assignAssessor,0)),
  assert(threadID(inspectCar,0)),
  assert(threadID(sendNewRepairCost,0)),
  assert(threadID(agreeRepair,0)),
  assert(threadID(repairCar,0)),
  assert(threadID(sendInvoice,0)),
  assert(threadID(forwardInvoice, 0)).


clear:-
 retract(absolute_deadline(phoneClaim,_,_)),
 retract(absolute_deadline(receiveInfo,_,_)),
 retract(absolute_deadline(assignGarage,_,_)),
```

```
retract(absolute_deadline(sendCar,_,_)),
retract(absolute_deadline(estimateRepairCost, _,_)),
retract(absolute_deadline(notifyClaim, _,_)),
retract(absolute_deadline(forwardClaim, _,_)),
retract(absolute_deadline(contactGarage,_,_)),
retract(absolute_deadline(sendClaimForm,_,_)),
retract(absolute_deadline(returnClaimForm,_,_)),
retract(absolute_deadline(sendRepairCost,_,_)),
retract(absolute_deadline(assignAssessor,_,_)),
retract(absolute_deadline(inspectCar,_,_)),
retract(absolute_deadline(sendNewRepairCost,_,_)),
retract(absolute_deadline(agreeRepair,_,_)),
retract(absolute_deadline(repairCar,_,_)),
retract(absolute_deadline(sendInvoice,_,_)),
retract(absolute_deadline(forwardInvoice, _,_)),
retract(absolute_deadline(payRepairCost,_,_)),

retract(perform_time(phoneClaim,_,_)),
retract(perform_time(receiveInfo,_,_)),
retract(perform_time(assignGarage,_,_)),
retract(perform_time(sendCar,_,_)),
retract(perform_time(estimateRepairCost, _,_)),
retract(perform_time(notifyClaim, _,_)),
retract(perform_time(forwardClaim, _,_)),
retract(perform_time(contactGarage,_,_)),
retract(perform_time(sendClaimForm,_,_)),
retract(perform_time(returnClaimForm,_,_)),
retract(perform_time(sendRepairCost,_,_)),
retract(perform_time(assignAssessor,_,_)),
retract(perform_time(inspectCar,_,_)),
retract(perform_time(sendNewRepairCost,_,_)),
retract(perform_time(agreeRepair,_,_)),
retract(perform_time(repairCar,_,_)),
retract(perform_time(sendInvoice,_,_)),
retract(perform_time(forwardInvoice, _,_)),
retract(perform_time(payRepairCost,_,_)),

retract(state(phoneClaim,_)),
retract(state(receiveInfo,_)),
retract(state(assignGarage,_)),
retract(state(sendCar,_)),
retract(state(estimateRepairCost, _)),
retract(state(notifyClaim, _)),
retract(state(forwardClaim, _)),
retract(state(contactGarage,_)),
retract(state(sendClaimForm,_)),
retract(state(returnClaimForm,_)),
retract(state(sendRepairCost,_)),
retract(state(assignAssessor,_)),
```

```
 retract(state(inspectCar,_)),
 retract(state(sendNewRepairCost,_)),
 retract(state(agreeRepair,_)),
 retract(state(repairCar,_)),
 retract(state(sendInvoice,_)),
 retract(state(forwardInvoice, _)),
 retract(state(payRepairCost,_)),

 retract(threadID(receiveInfo,_)),
 retract(threadID(assignGarage,_)),
 retract(threadID(sendCar,_)),
 retract(threadID(estimateRepairCost, _)),
 retract(threadID(notifyClaim, _)),
 retract(threadID(forwardClaim, _)),
 retract(threadID(contactGarage,_)),
 retract(threadID(sendClaimForm,_)),
 retract(threadID(returnClaimForm,_)),
 retract(threadID(sendRepairCost,_)),
 retract(threadID(assignAssessor ,_)),
 retract(threadID(inspectCar,_)),
 retract(threadID(sendNewRepairCost,_)),
 retract(threadID(agreeRepair,_)),
 retract(threadID(repairCar,_)),
 retract(threadID(sendInvoice,_)),
 retract(threadID(forwardInvoice, _)).


/*--------------- occur ---------------------*/
occur(A):- state(X,never) -> sender(X,S), write(S),
    write('␣violate␣contract,␣did␣not␣do␣'),
    write(X), write('.␣So␣all␣business␣processes␣stops!'),
    !, fail.

occur(A):- precondition(A)->
   sender(A,S),
   receiver(A,R),
   get_time(GT),
   convert_time(GT, Year, Month, Day, Hour, Min, Sec,_),
   retract(state(A,_)),
   assert(state(A,box)),
   retract(perform_time(A,0,0)),
   assert(perform_time(A,Hour,Min)),
   absolute_deadline(A,AH,AM),
   retract(absolute_deadline(A,AH,AM)),
   assert(absolute_deadline(A,null,null)),
   ((A == phoneClaim) -> !;
     ((A == payRepairCost) -> thread_send_message(main, exit);
           (not(threadID(A,_)) -> !; threadID(A,T),
     thread_send_message(T, occurred)))),
```

```
    write(S),
    write('␣has␣performed␣'), write(A),
    write('␣for␣'), write(R), write('␣at␣'),
    write(Year),write('/'),
    write(Month), write('/'),
    write(Day), write('␣'),
    write(Hour), write(':'),
    write(Min), nl,nl,
    c_action(C,A,Attribute),
    (Attribute == 'tr'-> (write('Commitment␣'), write(C),
        write('␣has␣been␣triggered!'),nl,nl);
      Attribute == 'in'-> (write('Commitment␣'), write(C),
        write('␣is␣processing!'),nl,nl);
      Attribute == 'fi'-> (write('Commitment␣'), write(C),
        write('␣is␣finished!'),nl,nl)),
     postcondition(A);
     !,fail.

/*---------- precondition ----------*/
precondition(A):-
  not((constraint(_X,(A|_)))|(constraint(_X,(_|A)))|
  (constraint(_X,A))).

precondition(A):-
  constraint(X,(A|_)) ->
    (state(X,ST),
    (ST == notyet| ST == diamond) ->
    reminding, sender(X,S),
    write(S), write('␣should␣occur␣'), write(X),
    write('␣before'),write('.'),nl,!,fail;
    (intime_all -> (state(X,ST),ST == box  -> true);
    !, fail)).

precondition(A):-
        constraint(X,(_|A)) ->
         (state(X,ST),
         (ST == notyet| ST == diamond)  ->
          reminding,  sender(X,S),
          write(S), write('␣should␣occur␣'),
          write(X), write('␣before'),
          write('.'),nl,!,fail;
          (intime_all -> (state(X,ST),ST == box -> true);
          !, fail)).

precondition(A):-
        constraint(X,A) ->
        (state(X,ST),
        (ST == notyet| ST == diamond) ->
         reminding, sender(X,S),
```

```
          write(S), write('␣should␣occur␣'),
          write(X), write('␣before'),
          write('.'),nl, !, fail;
          (intime_all -> (state(X,ST), ST == box  ->true);
          !, fail)).

precondition(A):-
         constraint((X|Y),A) ->
         (((state(X,ST1), state(Y,ST2),
         (ST1 == notyet|ST1 == diamond),
           ST2 == box) -> reminding, sender(X,S1),
            write(S1), write('␣should␣occur␣'),
            write(X), write('␣before'),
            write('.'), nl, !, fail;!),
           ((state(X,ST1), state(Y,ST2),ST1 == box,
            (ST2 == notyet| ST2 == diamond)) ->
            reminding, sender(Y,S2),
            write(S2), write('␣should␣occur␣'),
            write(Y), write('␣before'),
            write('.'), nl, !, fail;!),
           ((state(X,ST1), state(Y,ST2),
            (ST1 == notyet| ST1 == diamond),
            (ST2 == notyet| ST2 == diamond)) ->
              reminding, sender(X,S1), sender(Y,S2),
              write(S1), write('␣should␣occur␣'),
              write(X), write('␣before'),
              write('.'), nl,
              write(S2), write('␣should␣occur␣'),
              write(Y), write('␣before'),
              write('.'), nl,!, fail;!),
              (intime_all -> ((state(X,ST1), state(Y,ST2),
                 ST1 == box, ST2 == box ) ->true); !, fail)).


precondition(A):- state(A,ST),
       (ST == box -> write(A), write('␣has␣occured'),
        write('␣does␣not␣need␣to␣occur␣again'),nl).


/*--------- postcondition ----------*/

postcondition(A):-
  constraint((A|Z),Y)  ->
  (state(Z,ST), ST == box ->
  (sender(Z,S1), perform_time(A,PH1,PM1),
  write(S1), write('␣has␣performed␣'),
  write(Z), write('␣at.'), write(PH1),
  write(':'), write(PM1),nl,nl,
  retract(state(Y,notyet)),
```

```
    assert(state(Y,diamond)),
    sender(Y,S2), deadline(Y,D2),
    perform_time(Z,PH2,PM2),
    max(PH1,PM1,PH2,PM2,PH, PM),
    time_display(PH,PM,D2,TH,TM),
    retract(absolute_deadline(Y,_,_)),
    assert(absolute_deadline(Y,TH,TM)),
    write(S2), write(' would perform '), write(Y),
    write(' before '), write(TH), write(':'),
    write(TM), write('.'), nl);
    reminding,
    thread_create(warning(Y),ThreadID,[detached(true)]),
    retract(threadID(Y,_)),
    assert(threadID(Y,ThreadID)),
    sender(Z,S1),
    write(S1), write(' should perform '), write(Z),
    write(' before '),nl ).

postcondition(A):-
    constraint((Z|A),Y) ->
    (state(Z,ST), ST == box ->
    (sender(Z,S1), perform_time(A,PH1,PM1),
    write(S1), write(' has performed '),
    write(Z), write(' at.'),
    write(PH1), write(':'), write(PM1),nl,nl,
    retract(state(Y,notyet)),
    assert(state(Y,diamond)),
    sender(Y,S2), deadline(Y,D2),
    perform_time(Z,PH2,PM2),
    max(PH1,PM1,PH2,PM2,PH, PM),
    time_display(PH,PM,D2,TH,TM),
    retract(absolute_deadline(Y,_,_)),
    assert(absolute_deadline(Y,TH,TM)),
    write(S2), write(' would perform '),
    write(Y), write(' before '),
    write(TH), write(':'), write(TM),
    write('.'), nl);
    reminding,
    thread_create(warning(Y), ThreadID,[detached(true)]),
    retract(threadID(Y,_)),
    assert(threadID(Y,ThreadID)),
    sender(Z,S1),
    write(S1), write(' should perform '), write(Z),
    write(' before '), nl ).

postcondition(A):-
    constraint(A,Y) ->
    (retract(state(Y,notyet)), assert(state(Y,diamond)),
    sender(Y,S), deadline(Y,D), perform_time(A,PH,PM),
```

```
  time_display(PH,PM,D,TH,TM),
  retract(absolute_deadline(Y,_,_)),
  assert(absolute_deadline(Y,TH,TM)),
  thread_create(warning(Y), ThreadID,[detached(true)]),
  retract(threadID(Y,_)),
  assert(threadID(Y,ThreadID)),
  write(S), write('␣would␣process␣'),
  write(Y), write('␣next␣before␣'),
  write(TH), write(':'), write(TM),
  write('.'), nl, reminding).

postcondition(A):-
  constraint(A,(Y|Z))->
  (retract(state(Y,notyet)), assert(state(Y,diamond)),
  sender(Y,S1), deadline(Y,D1), perform_time(A,PH,PM),
  time_display(PH,PM,D1,TH1,TM1),
  retract(absolute_deadline(Y,_,_)),
  assert(absolute_deadline(Y,TH1,TM1)),
  thread_create(warning(Y), ThreadID1,[detached(true)]),
  retract(threadID(Y,_)),
  assert(threadID(Y,ThreadID1)),
  write(S1), write('␣would␣process␣'),
  write(Y), write('␣next␣before␣'),
  write(TH1), write(':'), write(TM1),write('.'), nl,
  retract(state(Z,notyet)), assert(state(Z,diamond)),
  sender(Z,S2), deadline(Z,D2),
  time_display(PH,PM,D2,TH2,TM2),
  retract(absolute_deadline(Z,_,_)),
  assert(absolute_deadline(Z,TH2,TM2)),
  thread_create(warning(Z), ThreadID2,[detached(true)]),
  retract(threadID(Z,_)),
  assert(threadID(Z,ThreadID2)),
  write(S2), write('␣would␣process␣'),
  write(Z), write('␣next␣before␣'),
  write(TH2), write(':'), write(TM2),
  write('.'), nl,
  reminding).

 postcondition(A):- not(constraint(A,_Y)).


/*---------- time display ----------*/

time_display(SH,SM,D,TH,TM) :-
              M is SM+D,
              M >= 60 -> (TH is SH +1,
                          TM is SM+D-60);
                          (TH is SH,
                           TM is SM+D).
```

```
intime_all:-
    absolute_deadline(phoneClaim,H1,M1),
    ((not(H1 == null), not(M1 == start)) ->
    compare_time(phoneClaim,H1,M1,Visited);true),
    absolute_deadline(receiveInfo,H2,M2),
    ((not(H2 == null), not(M2 == start)) ->
    compare_time(receiveInfo,H2,M2,Visited);true),
    absolute_deadline(assignGarage,H3,M3),
    ((not(H3 == null), not(M3 == start)) ->
    compare_time(assignGarage,H3,M3,Visited);true),
    absolute_deadline(sendCar,H4,M4),
    ((not(H4 == null), not(M4 == start)) ->
    compare_time(sendCar,H4,M4,Visited);true),
    absolute_deadline(estimateRepairCost,H5,M5),
    ((not(H5 == null), not(M5 == start)) ->
    compare_time(estimateRepairCost,H5,M5,Visited);true),
    absolute_deadline(notifyClaim,H6,M6),
    ((not(H6 == null), not(M6 == start)) ->
    compare_time(notifyClaim,H6,M6,Visited);true),
    absolute_deadline(forwardClaim,H7,M7),
    ((not(H7 == null), not(M7 == start)) ->
    compare_time(forwardClaim,H7,M7,Visited);true),
    absolute_deadline(contactGarage,H8,M8),
    ((not(H8 == null), not(M8 == start)) ->
    compare_time(contactGarage,H8,M8,Visited);true),
    absolute_deadline(sendClaimForm,H9,M9),
    ((not(H9 == null), not(M9 == start)) ->
    compare_time(sendClaimForm,H9,M9,Visited);true),
    absolute_deadline(returnClaimForm,H10,M10),
    ((not(H10 == null), not(M10 == start)) ->
    compare_time(returnClaimForm,H10,M10,Visited);true),
    absolute_deadline(sendRepairCost,H11,M11),
    ((not(H11 == null), not(M11 == start)) ->
    compare_time(sendRepairCost,H11,M11,Visited);true),
    absolute_deadline(agreeRepair,H12,M12),
    ((not(H12 == null), not(M12 == start)) ->
    compare_time(agreeRepair,H12,M12,Visited);true),
    absolute_deadline(repairCar,H13,M13),
    ((not(H13 == null), not(M13 == start)) ->
    compare_time(repairCar,H13,M13,Visited);true),
    absolute_deadline(sendInvoice,H14,M14),
    ((not(H14 == null), not(M14 == start)) ->
    compare_time(sendInvoice,H14,M14,Visited);true),
    absolute_deadline(forwardInvoice,H15,M15),
    ((not(H15 == null), not(M15 == start)) ->
    compare_time(forwardInvoice,H15,M15,Visited);true),
    absolute_deadline(payRepairCost,H16,M16),
```

```
    ((not(H16 == null), not(M16 == start)) ->
    compare_time(payRepairCost,H16,M16,Visited);true),
    not(Visited == 0) -> true; fail.

compare_time(X,H,M,Visited):-
        get_time(GT),
        convert_time(GT, _, _, _, Hour, Min, _, _),
        ((Hour > H ; Hour == H, Min > M)->
          retract(state(X,_)),
          assert(state(X,never)),
          write(X),
          write('␣out␣of␣time.'),
          write('␣All␣business␣processes␣stops!'),nl,
          (not(Visited == 0) -> Visited =0);true).

intime(X):-
        absolute_deadline(X,PH,PM),
        get_time(GT),
        convert_time(GT, _, _, _, Hour, Min, Sec, _),
        (Hour > PH ; Hour == PH, Min > PM;
                Hour == PH, Min == PM, Sec >59 )->
             retract(state(X,_)),
             assert(state(X,never)),
             write(X),
             write('␣out␣of␣time,'),
             write('␣All␣business␣processes␣stops!'),nl,
             !, fail; true.

/*-------------- max time --------------*/
max(PH1,PM1, PH2,PM2, PH, PM):-
    PH1 > PH2 ->(PH is PH1, PM is PM1);
      (PH1 == PH2 ->(PH is PH1,
             PM1 >= PM2 -> PM is PM1; PM is PM2);
             PH1 < PH2 -> (PH is PH2, PM is PM2)).

/*-------------- reminding --------------*/
reminding :-
        forall((state(X,diamond),intime(X)),
           (sender(X,S), absolute_deadline(X,PH,PM),
           write(S), write('␣would␣process␣'), write(X),
           write('␣before␣'), write(PH),write(':'),
           write(PM), nl)).


/*---------------- state of contract ----------*/
state_of_contract:-
    nl,
    write('=====␣State␣of␣contract␣process:␣======'),nl,
    state(phoneClaim, S1),
```

```
    state(receiveInfo, S2),
    state(assignGarage, S3),
    state(sendCar, S4),
    state(estimateRepairCost, S5),
    state(notifyClaim, S6),
    state(forwardClaim, S7),
    state(contactGarage, S8),
    state(sendClaimForm, S9),
    state(returnClaimForm, S10),
    state(sendRepairCost, S11),
    state(agreeRepair, S12),
    state(repairCar, S13),
    state(sendInvoice, S14),
    state(forwardInvoice, S15),
    state(payRepairCost, S16),

    ((S1 == 'box', S2 == 'box', S3 == 'box', S6 == 'box')
    ->
    write('Commitment phoneService
    has be finished!'),
    nl;
    (state(phoneClaim, S1),
    state(receiveInfo, S2),
    state(assignGarage, S3),
    state(notifyClaim, S6),
    (S1 == 'notyet'; S1 == 'diamond'),
    (S2 == 'notyet'; S2 == 'diamond'),
    (S3 == 'notyet'; S3 == 'diamond'),
    (S6 == 'notyet'; S6 =='diamond'))
      ->
     write('Commitment phoneService
    has not be started yet!'),
     nl;
     write('Commitment phoneService
    is processing!'),nl),

    ((S4 == 'box', S5 == 'box',
            S12 == 'box', S13 == 'box')
     ->
    write('Commitment repairService
    has be finished!'),
    nl
    ;
    (state(sendCar, S4),
    state(estimateRepairCost, S5),
    state(agreeRepair, S12),
    state(repairCar, S13),
    (S4 == 'notyet'; S4 == 'diamond'),
    (S5 == 'notyet'; S5 == 'diamond'),
```

```
    (S12 == 'notyet'; S12 == 'diamond'),
    (S13 == 'notyet'; S13 == 'diamond'))
         ->
    write('Commitment repairService has
    not be started yet!'),
    nl
    ;
    write('Commitment repairService
    is processing!'),
    nl),

    ((S6 == 'box', S9 == 'box', S10 == 'box')
     ->
    write('Commitment claimForm has
    be finished!'),
    nl
    ;
    (state(notifyClaim, S6),
    state(sendClaimForm, S9),
    state(returnClaimForm, S10),
    (S6 == 'notyet'; S6 == 'diamond'),
    (S9 == 'notyet'; S9 == 'diamond'),
    (S10 == 'notyet'; S10 == 'diamond'))
      ->
     write('Commitment claimForm has not
    be started yet!'),
      nl
      ;
      write('Commitment claimForm is processing!'),
      nl),

      ((S7 == 'box', S8 == 'box', S11 == 'box',
        S12 == 'box', S13 == 'box', S14 == 'box',
        S15 == 'box')
       ->
    write('Commitment dailyService has
    be finished!'),
    nl
    ;
    (state(forwardClaim, S7),
    state(contactGarage, S8),
    state(sendRepairCost, S11),
    state(agreeRepair, S12),
    state(repairCar, S13),
    state(sendInvoice, S14),
    state(forwardInvoice, S15),
    (S7 == 'notyet'; S7 == 'diamond'),
    (S8 == 'notyet'; S8 == 'diamond'),
    (S11 == 'notyet'; S11 == 'diamond'),
```

```
    (S12 == 'notyet'; S12 == 'diamond'),
    (S13 == 'notyet'; S13 == 'diamond'),
    (S14 == 'notyet'; S14 == 'diamond')),
    (S15 == 'notyet'; S15 == 'diamond')
      ->
     write('Commitment dailyService has not be
     started yet!'),
     nl
     ;
     write('Commitment dailyService is
     processing!'),nl),

    ((S10 == 'box', S15 == 'box', S16 == 'box')
      ->
     write('Commitment paymentService has be finished!'),
     nl
     ;
     (state(returnClaimForm, S10),
     state(forwardInvoice, S15),
     state(payRepairCost, S16),
     (S10 == 'notyet' ; S10 == 'diamond'),
     (S15 == 'notyet'; S15== 'diamond'),
     (S16 == 'notyet'; S16 == 'diamond'))
        ->
      write('Commitment paymentService has
      not be started yet!'),
      nl
      ;
      write('Commitment paymentService
      is processing!'),
      nl).

responsibility:-
    forall((state(A,never);state(A,diamond)),
      (forall(c_action(C,A,_),
         (write('Commitment '), write(C),
         write(' is violated!'),nl,nl)),
       forall((state(B,diamond),intime(B)),
         (sender(B,S), write(S),
         write(' pleases stop to do action '),
         write(B),
         write(', because a violation is found! '),
         nl,
         state(A,never), sender(A,S1), write(S1),
         write(' did not do action '),
         write(A), nl))
        )).
```

```
/*------------ warning ----------------*/
warning(X) :-
        deadline(X,PM),
        Y is PM*60 - 40,
        sleep(Y),
        not(thread_peek_message(occurred)),
        sender(X,S),
        nl, write(S), write('␣should␣perform␣'),
        write(X),
        write('␣within␣20␣seconds!!!␣␣'), nl,
        retract(threadID(X,_)),
        assert(threadID(X,warning_message_sent)),
        thread_join(X,_).


tt(0):-init.

tt(N) :-
    N > 0,
    N1 is N-1,
    tt(N1),
    set,
    occur(phoneClaim),
    occur(receiveInfo),
    occur(assignGarage),
    occur(notifyClaim),
    occur(sendCar),
    occur(forwardClaim),
    occur(sendClaimForm),
    occur(contactGarage),
    occur(estimateRepairCost),
    occur(sendRepairCost),
    occur(assignAssessor),
    occur(inspectCar),
    occur(sendNewRepairCost),
    occur(agreeRepair),
    occur(repairCar),
    occur(returnClaimForm),
    occur(sendInvoice),
    occur(forwardInvoice),
    occur(payRepairCost),
    clear.


io(N):-
    tell('t1.txt'),
    write(N),nl,
    get_time(GT),
    convert_time(GT, _, _, _, Hour, Min, Sec, Minsec),
```

```
    write('Start time: '), write(Hour),
    write(':'), write(Min),
    write(':'), write(Sec),
    write(':'), write(Minsec),nl,
    told,
    tt(N),
    tell('t2.txt'),
    get_time(GT1),
    convert_time(GT1, _, _, _, Hour1, Min1, Sec1, Minsec1),
    write('End time: '), write(Hour1),
    write(':'), write(Min1),write(':'),
    write(Sec1),write(':'),
    write(Minsec1),nl,
    G is GT1-GT,
    convert_time(G,_,_,_,H,M,S,Ms),
    write('between time: '), write(H),
    write(':'), write(M),write(':'),
    write(S),write(':'), write(Ms),nl,
    told.


occur_pure(A):-
  precondition(A)->
  (sender(A,S),
  receiver(A,R),
  get_time(GT),
  convert_time(GT, Year, Month, Day, Hour, Min, _,_),
  retract(state(A,_)),
  assert(state(A,box)),
  retract(perform_time(A,0,0)),
  assert(perform_time(A,Hour,Min)),
  absolute_deadline(A,AH,AM),
  retract(absolute_deadline(A,AH,AM)),
  assert(absolute_deadline(A,null,null)),nl,
  write(S),
  write(' has performed '), write(A),
  write(' for '), write(R), write(' at '),
  write(Year),write('/'),
  write(Month), write('/'),
  write(Day), write(' '),
  write(Hour), write(':'),
  write(Min),  nl,
  c_action(C,A,Attribute),
   (Attribute == 'tr'-> (write('Commitment '),
    write(C), write(' has been triggered!'),
    nl,nl);
      Attribute == 'in'-> (write('Commitment '),
      write(C), write(' is processing!'),nl,nl);
       Attribute == 'fi'-> (write('Commitment '),
```

```
        write(C), write('␣is␣finished!'),nl,nl))).


tt_pure(0):-init.

tt_pure(N) :-
    N > 0,
    N1 is N-1,
    tt_pure(N1),set,
    occur_pure(phoneClaim),
    occur_pure(receiveInfo),
    occur_pure(assignGarage),
    occur_pure(notifyClaim),
    occur_pure(sendCar),
    occur_pure(forwardClaim),
    occur_pure(sendClaimForm),
    occur_pure(contactGarage),
    occur_pure(estimateRepairCost),
    occur_pure(sendRepairCost),
    occur_pure(assignAssessor),
    occur_pure(inspectCar),
    occur_pure(sendNewRepairCost),
    occur_pure(agreeRepair),
    occur_pure(repairCar),
    occur_pure(returnClaimForm),
    occur_pure(sendInvoice),
    occur_pure(forwardInvoice),
    occur_pure(payRepairCost),clear.

io_pure(N):-
    tell('t3.txt'),
    write(N),nl,
    get_time(GT3),
    convert_time(GT3, _, _, _, Hour3, Min3, Sec3, Minsec3),
    write('Start␣time:␣'), write(Hour3),
    write(':'), write(Min3),
    write(':'), write(Sec3),
    write(':'), write(Minsec3),nl,
    told,
    tt_pure(N),
    tell('t4.txt'),
    get_time(GT4),
    convert_time(GT4, _, _, _, Hour4, Min4, Sec4, Minsec4),
    write('End␣time:␣'), write(Hour4),
    write(':'), write(Min4),write(':'),
    write(Sec4),write(':'),
    write(Minsec4),nl,
    G1 is GT4 - GT3,
    convert_time(G1, _, _, _, H2,M2,S2,Ms2),
```

```
write('between time: '), write(H2),
write(':'), write(M2),write(':'),
write(S2),write(':'), write(Ms2),nl,
told.
```

Appendix C

Parameters

Parameters
In our experiments, we controlled the number of actions involved in a contract execution.
Experiment set up: hardware and software
Table C provides the detailed hardware and software:

CPU	Memory	Operation System	Prolog
1400HZ AMD Athlon	256MB	Microsoft Windows 2000 professional	SWI-prolog(Multi-threaded, Version 5.1.9) [swi]

Table C.1: Software and hardware specifications for experiments